NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

COST ESTIMATION OF POST PRODUCTION SOFTWARE SUPPORT IN GROUND COMBAT SYSTEMS

by

Christopher J. Cannon

September 2007

Thesis Co-Advisors: Daniel Nussbaum
Gregory K. Mislick

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 2007	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE Cost Estimation of Post Production Software Support in Ground Combat Systems	5. FUNDING NUMBERS
6. AUTHOR(S) Captain Christopher J. Cannon	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE A

13. ABSTRACT (maximum 200 words)

Weapon systems and programs are becoming increasingly more dependent on software as a critical technology for the success of the programs. Along with this dependence on performance, the costs associated with software are becoming an increasing share of the life cycle costs of these weapon systems and programs. Life cycle software costs are divided into two phases, development and maintenance. There are numerous popular models to aid developers and independent estimators in predicting costs and schedules for software development. Some of these models are open source, many others are proprietary. These models are based on research performed on existing software systems and historical data.

However, for software maintenance, there are far fewer models, research efforts, or collected data sets. The Army's term for software maintenance is post production software support. This thesis describes how this support is currently funded, performed, and estimated. The model presented could be adopted to manage support of Army ground combat systems.

This thesis is furthers the understanding of the software maintenance support costs associated with weapon systems. In addition to specific results on ground combat systems presented, the thesis provides insight into maintaining other large software-dependent systems and recommendations on further research in the field.

14. SUBJECT TERMS Cost Estimation, Post Production Software Support, Future Combat System, 3-parameter Gamma Distribution, System Evaluation and Estimation of Resources – Software Estimation Model (SEER-SEM)	15. NUMBER OF PAGES 91
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

COST ESTIMATION OF POST PRODUCTION SOFTWARE SUPPORT IN GROUND COMBAT SYSTEMS

Christopher J. Cannon
Captain, United States Marine Corps
B.S., Carnegie Mellon University, 1998

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 2007

Author: Christopher J. Cannon

Approved by: Dr. Daniel Nussbaum
Thesis Co-Advisor

Gregory K. Mislick
Thesis Co-Advisor

James N. Eagle
Chairman, Operations Research Department

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Weapon systems and programs are becoming increasingly more dependent on software as a critical technology for the success of the programs. Along with this dependence on performance, the costs associated with software are becoming an increasing share of the life cycle costs of these weapon systems and programs. Life cycle software costs are divided into two phases, development and maintenance. There are numerous popular models to aid developers and independent estimators in predicting costs and schedules for software development. Some of these models are open source, many others are proprietary. These models are based on research performed on existing software systems and historical data.

However, for software maintenance, there are far fewer models, research efforts, or collected data sets. The Army's term for software maintenance is post production software support. This thesis describes how this support is currently funded, performed, and estimated. The model presented could be adopted to manage support of Army ground combat systems.

The major contribution of this thesis is furthering the understanding of the software maintenance support costs associated with weapon systems. In addition to specific results on ground combat systems presented, the thesis provides insight into maintaining other large software-dependent systems and recommendations on further research in the field.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS ACRONYMS

AFTOC	Air Force Total Ownership Costs
ATGM	Anti Tank Guided Missile
ARDEC	Armament Research Development and Engineering Center
CAIG	Cost Analysis Improvement Group
CMM	Capability Maturity Model
COCOMO	Constructive Cost Model
COTS	Commercial Off The Shelf
CY05$	Constant Year 2005 dollars
FCS	Future Combat System
GAO	Government Accountability Office
LSI	Lead Systems Integrator
LCC	Life Cycle Cost
LAV	Light Armored Vehicle
MGV	Manned Ground Vehicle
MATREX	Modeling Architecture for Technology, Research, and Experimentation
NBC	Nuclear, Biological, Chemical
NCCA	Naval Center for Cost Analysis
OSD	Office of the Secretary of Defense
OSMIS	Operating and Support Management Information Systems
OMA	Operation and Maintenance
OPN	Other Procurement Navy
PA&E	Programs Analysis and Evaluation
PEO	Program Execution Office
POM	Program Office Memorandum

REVIC	Revised Enhanced Version of Intermediate COCOMO
SEI	Software Engineering Institute
SLOC	Source Lines of Code
SME	Subject Matter Expert
SEER-SEM	System Evaluation and Estimation of Resources – Software Estimating Model
SoSCOE	System of Systems Common Operating Environment
TARDEC	Tank Automotive Research, Development, and Engineering Center
TACOM	Tank-Automotive Command
UAV	Unmanned Air Vehicle
UGS	Unmanned Ground Sensor
UGV	Unmanned Ground Vehicle
VAMOSC	Visibility and Management of Operating and Support Costs
VBA	Visual Basic for Applications

EXECUTIVE SUMMARY

The cost to maintain software is largely unknown. Historically, Post Production Software Support (PPSS) costs are estimated using simple scalars and analogies to previous systems. This thesis explores the relationship between the size of a software project and the cost of maintaining the completed software over the lifetime of the project. A data set on Army ground combat systems is analyzed. The data set contains PPSS budgets contained in the years 2002-2013.

This thesis researches appropriate variables for analysis, interprets the data, and comments on methodology of the analysis involved, as well as the allocated budgets. The analysis portion conducts t-tests to make inferences about similarities in the dataset, uses modeling software to determine the validity of the data, describes the data's distribution, and suggests useful metrics to make comparisons between years represented in the data set. In the absence of statistical evidence, a few reasonable assumptions are made based on knowledge obtained from relevant subject matter experts.

It is difficult to make valid suggestions about what the data represents for individual weapon systems until the budgeted years are finalized and executed. However, this thesis contains three main conclusions. First, that there is no statistical difference in the total annual resources (adjusted for inflation) budgeted for software maintenance on ground combat vehicles. This means that there appears to be a set annual budget for each program. The second observation is that annual amounts spent on software maintenance for ground combat vehicles follow a simple linear regression. As expressed in the data, there is an estimated growth of $2.95 million dollars a year, or a 5.06% rate of growth. The third observation suggests that the more expensive software programs supporting ground combat systems are getting an increasing share of the total budget, leaving the smaller programs fighting for smaller shares of the allocations.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Many people have contributed to this thesis. I would like thank my advisors, Greg Mislick, LtCol, USMC (Ret) whose support started the process and Dr. Dan Nussbaum, a treasure as a professor and a person. Thanks to Walter Cooper, OSD-PA&E, whose contacts, expertise, and guidance enabled my Pentagon visit to be a very productive experience. Thanks to John Thurman, LtCol, USA, who hosted my stay and Corinne Wallshein, who was working on her dissertation on software lifecycle costs and was extremely forthright in sharing her work. Thanks to Matt Kastantin, Curtis Khol, and John McCrillis, all of OSD-PA&E, for their collective wisdom. Thanks to James Judy, Chief of C4ISR Costing Division for the Army, Brian Fersch, and Ed Lesnowicz, for their thoughts on this difficult problem.

I'd also like to thank Ed Andres, Bryan Dunbar, and Phil Smith for taking the time to share their experience on engineering software for the Abrams, Bradley, and Stryker weapons systems. Lastly, I thank my parents Joyce Cannon, a teacher who still has lessons for me, and Dr. Thomas Cannon, Jr., who is among many things, an inspiration.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

> Software not developed with maintenance in mind can end up so poorly designed and documented that total redevelopment is actually cheaper than maintaining the original code.
>
> — U.S. Air Force's Software Technology Support Center, "Guidelines for Successful Acquisition and Management of Software-Intensive Systems" version 3.0 May 2000 .

A. THESIS OUTLINE

This thesis will explore and determine the relationship between the size of a software project and the cost of maintaining the completed software over the lifetime of the project. This thesis researches appropriate variables for analysis, interprets the data, and comments on methodology of the analysis involved, as well as the allocated budgets. The analysis portion conducts t-tests to make inferences about similarities in the dataset, compares metrics between years represented in the data set, and describes the data's distribution. In the absence of statistical evidence, a few reasonable assumptions are made based on knowledge obtained from relevant subject matter experts (SMEs). The findings suggest what the data means to budget managers. Lastly, conclusions are summarized and specific recommendations are discussed for further research.

B. OBJECTIVE

The objective of cost estimation is to collect and analyze historical data, apply quantitative models, techniques, tools, and databases, to predict the future cost of an item, program, or task. The specific purpose of this thesis is to conduct initial research in the costs associated with post production software support. Available data of current software support is analyzed, an interpretation is made of what the data represents, and recommendations are provided for policies and practices to obtain better data in order for more substantial research to be performed. From the data available, a model is presented to estimate the costs of software support for follow on systems in the same domain.

C. BACKGROUND

The development of software has become an increasingly important and expensive part of all major Department of Defense (DoD) acquisitions. Additionally, the complete Life Cycle Cost (LCC) of software is becoming a larger share of the project's total budget. Therefore, the importance of managing the ongoing software development and maintenance and the capability to calculate the costs of software maintenance is becoming increasingly important.

There are key differences between hardware and software systems. These differences are in their development cycles, supporting phases of management, and how the risks of each type of system are managed. One important aspect of software development is that software development delays cannot be mitigated by more money or resources. In historical examples, a software program that falls behind in development will never catch up. In fact, adding developers to a software program behind schedule has been demonstrated to actually prolong development time [1].

There has been much research on the costs to develop software. There are many models, open source and proprietary, available to help project and plan the development process. In comparison there are few tools to help project and plan the supporting effort needed to maintain software once it has already been fielded. When problems arise in the weapon system's functionality, warfighters submit requests for changes to the software known as "hotfixes." Software support facilities address the warfighters' needs in the order requested, providing new releases of software to enhance usability. When underfunded, these facilities adjust requests and priorities in order to fix the problems that their budgets allow.

To understand why these risks are occurring in software, it is important to define the exact problem. Why are program managers failing to account for risks associated with software? The situation is concisely surmised in the Navy's Open Architecture Computing Environment Design Guidance Version, published in 2004.

> A major characteristic of today's computing industry is the fact that the technology base is changing and evolving rapidly. This potentially provides great benefit in terms of a steadily improving price/performance ratio. However, systems that are not designed to accommodate this rapid

> change of the technology base can be quite costly to maintain over the life cycle. Application source code not designed for portability should be modified, sometimes extensively, when it is ported to new networks, computers, operating systems, middleware, etc. In some cases, the changes can be so extensive that redesign of the system may be necessary.
>
> Conversely, through proper use of standards and isolation layers that hide implementation details, it is possible to design components so that application source code can be ported across a wide variety of underlying computing technologies. Achieving this objective requires exercising sufficient design and implementation discipline to forego use of vendor-unique features. This is sometimes difficult since vendor-unique features may confer a modest or even substantial advantage in performance or in initial cost; however, the long-term cost of repeated use of non-portable source code to a succession of technology bases often eventually far exceeds any initial cost savings or performance gains [2].

The Navy defines adaptability, portability, and discipline in design and implementation as the major characteristics of cost control. The Navy's guide is comprehensive, but does not provide analysis of the current software environment. What happens when a software program isn't portable or the implementation isn't disciplined? Without a proper perspective it is easy to dismiss software development as non-critical issues in a program's development. Three recent software projects, two failures and one success, provide insight on the importance of quality software. The Denver airport originally scheduled to open in 1993 was delayed over 18 months at a cost of over $190 million due to software and automation delays. In 1996, the European Ariane 5 rocket on its maiden flight was destroyed due to a software error, carrying $500 million in payload. In contrast, the developers that maintain the on-board code for NASA Space Shuttle experience an error rate in their code orders of magnitude less than an equivalent commercial center. More details are available on these examples in Appendix A.

Most software support for DoD's weapon systems is performed at Software Engineering Centers (SEC). These centers produce technical documentation and data for software systems. Historically they release a new version every 18 months. In reality, they also serve as one stop shopping for many software needs dictated by their parent organizations. This may range from providing training or IT services to writing web pages. They provide the logistics support to ensure weapons systems are uploaded with

new software. They field requests and perform maintenance typically viewed as corrective, defective, adaptive, or enhancement. There may be problems with the original software, or they may correct hardware deficiencies that are more efficiently addressed with software solutions. They perform regression testing, testing the existing codes' functionality over the range of operation, on each change they make. This testing is performed to prevent the type of errors that led to the Adriane 5 disaster. Specifications detailing the tests to be performed may run thousands of pages [3].

To discern what it costs to run software on a weapon system, it's important to look at the base case. What does it cost to run any weapon program? Typically long term planners such as an oversight committee or programming office review the full life cycle cost for a particular system. Life cycle costs are broken down into stages. For hardware, the DoD 5000 acquisition model identifies these separate stages: research and development, production, and operation and support. But for software, the phases are development, post development software support, and post production software support. In determining the total LCC, analysts use different estimation techniques to project the costs in each of these phases.

For hardware items such as naval vessels, aircraft, and ground combat vehicles, cost estimating tools have been developed over time with historical precedence. Typically one of four techniques is employed to arrive at an estimate. The most accurate estimates are made from extrapolations of actual data. A recent Government Accountability Office (GAO) estimate calculated the cost for a new F-22 Raptor at $166 million [4]. This result is based upon real production data from the previous years. For this platform in particular, adjustments were made for changes being performed to strengthen the airframe. In a similar manner, the aircraft's operation and support costs can be calculated from available data. Extrapolation from actuals is the preferred technique because it produces good estimates (low variance) and analysts can present their figures to decision makers with a high degree of confidence.

A second technique, parametric estimates, is utilized for less mature programs. The estimate is based on known physical and performance characteristics of the item in question. This is applicable on many military systems such as missiles where actual

production costs have been observed on properties such as length, diameter, weight, and thrust. Parametric estimates are attractive because they can be performed quickly and are relatively easy to interpret for anyone with a statistical or engineering background.

A third estimation technique is known as engineering build up. This method attempts to find the sum of the costs from the breakdown of all work performed in the construction of a hardware item. Of all cost estimation techniques, engineering build up is the most expensive and the most time consuming to produce. However it is preferred for some systems such as naval vessels, where the historical data on work breakdowns is rich and each hardware unit is unique or may be constructed during unique circumstances.

In the absence of quality data to produce extrapolations, a parametric estimate, or build up the cost from its separate parts, the fourth technique is to make an analogy to previous weapon systems. For example, the F-35 Lightning II has a high degree of commonality with the F-22 Raptor. Many of the life cycle costs for the F-35 have been based on the F-22 [5]. In fact, this was a contributing factor in awarding the contract to Lockheed. Additionally, estimates for the Stryker, originally planned as an interim solution, were based on previous versions of the LAV.

Historically, cost estimation of software development has proven to be a difficult problem. There are numerous models used for software cost estimation, but they have many similarities. Well known models base their estimates using three basic steps.

- First they attempt to define the project's size.
- Next they aim at determining the software developers' productivity; which has a high degree of variability and is generally a function of available programmers.
- Lastly, estimators attempt to determine how much a programmer costs.

When trying to correctly size a program, most models approximate project size by counting the Source Lines of Code (SLOC). A SLOC is an executable statement as defined in the context of the language it was written in. Blank lines, comment lines, or lines with simply an "if" statement or an open bracket would not count.

Some models try to break down a project into a number of function points (FP). An FP is defined as one of five types of simple function operations: input, output,

inquiry, file, or interface operation. Essentially the idea is the same; how big is the program in basic building blocks?

Analysis of historical projects has show that SLOC is not by itself a good indicator of project length. One reason is that no developer writes a software program using entirely new code. In some cases it may be faster and more efficient to incorporate preexisting code. Sometimes code is divided into different categories that define the level of effort required to write it. Such code is termed "redesigned," "reimplemented," or "retest." Redesigned code is utilized that was created elsewhere, but requires some reengineering or test work.

$$SLOC = NewSize + ExistingSize \times (0.4 \times Redesign + 0.25 \times Reimpl + 0.35 \times Retest)$$

Reimplemented code is from previous versions, rewritten to correspond to new data requirements. The "retest" code may simply be the amount of code that undergoes regression testing. The contractor may use purchased code known as Commercial Off The Shelf (COTS) software. This additionally implies the need to write "glue" code which simply incorporates COTS or other code within the final product. Finally there will be some actual new code that was developed from scratch. As might be expected, different predictive models end up using different labeling schemes, such as "organic," "embedded," or "semi-detached."

Typically software estimation models use simple linear formula to determine total SLOC. For maintenance estimates, similar methodologies are used. Some analysts in the cost estimation community do not like this approach. The data that these models base their estimations upon may be small, from a unique domain, very different, or very proprietary. The suspicion is that the models are over-fit and that with simple manipulation users can produce whatever estimates they desire. For proper estimates, analysts prefer developing models based on available data, rather than relying on empirical scalars drawn from data that is unavailable to them.

Once the system has been appropriately sized, estimation models then make determinations on the software's complexity and the developer's effectiveness to determine productivity. Effectiveness is largely determined by the Capability Maturity

Model (CMM) score, a widespread metric developed by Carnegie Mellon University's Software Engineering Institute (SEI). SEI was established in 1984 as a federally funded research and development center. CMM scoring is on a five point scale. Rather than just an evaluation, CMM is an ongoing process that has been proven to substantially increase productivity of software development. Pioneered in the early 1990s, currently all software companies competing for DoD contracts will have a CMM score in order to place a bid. For complex systems such as FCS, a CMM rating of three was determined to be the minimum in order to simply place a bid. Estimating their own software development time is a mandatory quality assurance step contractors must accomplish in order to initially receive any type of CMM score.

Using these rough parameters for size and productivity, the algorithms perform a head-count on the number of developers and determine the duration and cost for the software development. System Evaluation and Estimation of Resources - Software Estimating Model (SEER-SEM), is the most popular proprietary model in use today. The main advantage SEER-SEM has over other estimators is that the model comes with a proprietary database of thousands of completed software projects. From an analyst's perspective, the main difference between estimating software development and software support is that there are virtually no models, databases, or research efforts on which software support models can be built.

The first widely used estimator was the Cost Constructive Model (COCOMO), designed by Dr. Barry Boehm in 1981. The purpose of the model was to predict the number of man months necessary to complete software development. The model was constructed on data from 60 projects at TRW ranging from 20,000 to 100,000 SLOC. The COCOMO model has been updated and is still in wide use today. While the COCOMO model was better than any previous tool, the accuracy of the results were was poor. A study using early versions of COCOMO found that average error was over 600 percent between predicted and actual development effort [6]. High estimation error is not necessarily an indicator of poor estimation tools. Errors may be due to high estimation complexity and insufficient cost control within the program. But clearly 600 percent average error is not an acceptable standard! It has been suggested that an acceptable

model would produce predicted costs within 25 percent of actual costs 75 percent of the time [7]. The most risky software programs are usually large, complex, incorporate new technology, or require additional integration from subcomponents. In all phases, especially in regard to software development, FCS exhibits each of these properties. A brief synopsis of FCS development helps to explain the importance of good software design.

The single largest DoD project under development is the Army's FCS. Its LCC is estimated to exceed the LCCE of the Joint Strike Fighter. The requirements and number of platforms continue to change, but in late 2006 the program involved new development of 9 Manned Ground Vehicles (MGVs) mostly from the same chassis, 6 Unmanned Ground Vehicles (UGVs), 5 Unmanned Air Vehicles (UAVs), and 4 Unmanned Ground Sensors (UGSs). In addition to over 20 vehicles and sensors being developed with their associated software, FCS requires an overarching architecture for communication and integration called the System of Systems Common Operating Environment (SoSCOE).

One indicator of software complexity is the wide range of software domains to consider. Current estimation models use different coefficients corresponding to different types of software code (user interfaces, data-link software, crypto, UAV software, ground manned and unmanned systems software, etc.) to estimate the final SLOC. The F-22 software development effort is metered out to 20 different contractors. Contractors have learned that SLOC equates to money. Often contractors maintain high estimates in order to pay for overheads that do not directly contribute to the contract. When analyzing the software support efforts that are performed by DoD it is crucial to focus on the largest programs. There are numerous reasons why this is the case. Like many other acquisition activities, the smaller software programs are often better run in terms of costs, performance, and schedule. Large software efforts invariably contain all the risky cost multipliers of complexity, growth, and integration. Lastly, it is the large programs that need more software support, while for smaller programs the delivered code may not have any requests for changes or releases of new versions.

In 2006, Army Chief of Staff General Schoomaker insisted that the entire FCS program stay within its $120 billion budget [8], yet the 2006 CAIG estimate places FCS

development at $300 billion. Estimates for the LCC continue to grow and the schedule continues to slide. A 2004 GAO report surmises the difficulties in FCS development:

> At conflict are the program's unprecedented technical challenges and time. At a top level, the technical challenges are: development of a first-of-a-kind network, 18 advanced systems, 53 critical technologies, 157 complementary systems, and 34 million lines of software code [9].

The program's complexity is also displayed by the amount of subcontractors involved in developing software for FCS, listed in Appendix B.

Determining who is writing the software is an on-going challenge, just as determining the level of effort required to field FCS. As early as 2003, according to both Congressional and CAIG sources, the software portion of the FCS was estimated at totaling between 32 and 33.7 million SLOC. There are no software programs that are close to the unprecedented size of FCS. By comparison, the Navy's Aegis system, Air Force's entire F/A-22 program, and NASA's International Space Station, each run at about 4.1 million SLOC. This makes the total FCS software development effort described in the Cost Analysis Requirement Description (CARD) ten times larger than any software program the U.S. government has ever fielded. The F-22's software design problems are detailed in Appendix C for comparison. Past CAIG estimates have priced FCS software development at $7-9 billion, with the software support at around $12.5 billion ($500 million a year). Given this variance in attempting to estimate production costs, developing accurate estimates for maintenance LCC remains a major challenge.

THIS PAGE INTENTIONALLY LEFT BLANK

II. DATA AND METHODOLOGY

A. DATA SET

A database on Post Production Software Support (PPSS) of weapon systems under the Army's Tank-Automotive Command (TACOM) was provided by the Army's G-4, DALO-RIL (Resource Integration Division), The Pentagon. The database was acquired for this thesis by Mr. Walt Cooper, a CAIG analyst with the Office of the Secretary of Defense, Programs Analysis and Evaluation (OSD PA&E). It is crucial to understand that this database represents funding requirements/projections. It is not data of actual expenditures spent on the associated systems, which is largely untraceable with the data collection efforts currently employed. Prior to entering this phase of the life cycle, the software developers for each system have had largely unique funding support. They have had different priorities from their program managers, fallen under different program executive offices, and may have had multiple sources of funding from supplementals, based on their contract terms, or other contingencies.

The data was contained on a Microsoft Excel spreadsheet, and needed considerable normalizing to be useful. Program Office Memorandum (POM) and OP29 figures were represented as individual sheets. Row entries corresponded to particular software programs. Column entries contain may fields of varying importance, most of which contained categorical data. The most vital data fields delineated the resources requested by the program, the amount funded (planned, not actual), which POM or OP29 of the request, year of the request, EINOMEN (a word description of the system), and a PRON (a unique identifier containing the year and weapons system information).

For the associated weapon systems found in the TACOM database, density data (i.e., the number of units of the system) was obtained from the Army's Operating and Support Management Information System (OSMIS). OSMIS density figures were configured to annual quantities. Annual figures were obtained for the Abrams Main Battle Tank (M1A1s, M1A2s), Bradley Fighting Vehicle Systems (M2A1, M2A2, M3A2), Paladins (M109A6), Strykers, and Wolverines (XM104). OSMIS allows for a

variety of options of data retrieval. For these purposes, the total number of vehicles in service in a given year was retrieved by querying over the major combatant commands and summing the results. These data were used to formulate hypotheses of whether the number of vehicles in service had an effect on programmed software maintenance costs.

B. VARIABLE SELECTION AND VALIDATION

The TACOM PPSS database contains 49 columns of data. The bulk of analysis was performed on the columnar data sets, namely on the planned funding amounts, year of request, and weapon system that each software project mapped to. The validity of discarding these columns was analyzed using SPSS (Statistical Package for the Social Sciences) Clementine version 11.1. SPSS was chosen because it handles categorical data well, while other software tools require transformations. Thirty-one columns displayed singular values or contained anomalous entries such as the username of the person entering the data, or a program identifier unique for each data point. These columns could not be included when Clementine was run to fit a model to the data, because they contained no real information. Based on inspection, it appeared that these remaining columns would not be statistically useful in predicting amounts requested or funded for software programs.

Three approaches were taken to determine if the remaining columns contained valid information. The results of the three approaches were compared.

- First, a regression tree was run in SPSS, to determine if the column data could predict whether the amount requested for a program fell above or below the median for the data set. Tree models are useful to describe some data sets because they are simple to construct and readily interpretable.
- More powerful statistical methods were used to determine if there was useful information contained in these apparently inadequate columns of data. We used a neural network since they are well implemented in SPSS and can quickly construct good models on difficult data sets where other modeling methods fail. The weakness of neural networks is that they are difficult to interpret, especially as the number of parameters and layers increase.

- Third, a model was created in SPSS from a logistic regression. The regression was run to predict if each program's requested monetary amount would fall above or below the population median.

Regression trees can create very flexible models but are susceptible to over-fitting. The model starts with all data points in a single node. It is an iterative process where the data is split into successive binary branches. The branching algorithm will attempt to partition the data using every possible binary split on every field. Typically, the algorithm chooses to split the data into two parts, minimizing the sum of squared deviations from the mean in the separate parts. Numerically, in each resulting node the algorithm minimizes deviance:

$$\sum \left(y - \bar{y} \right)^2$$

The algorithm proceeds until a sensible stopping point is reached, or splits can no longer be made.

The neural network was constructed to make the same prediction that was made with the tree model and logistic regression: whether the program's requested amount fell above or below the mean. Neural networks are implemented in SPSS using 3 types of nodes: inputs, outputs, and hidden nodes. The network consists of two sets of weighted arcs; one set from every input node to every hidden node and a second set from every hidden node to every output node. The value of any observation is determined by a three step process, as depicted in Figure 1. First, the product of the input node-hidden arc weights are summed. Next, an "activation function" is applied.

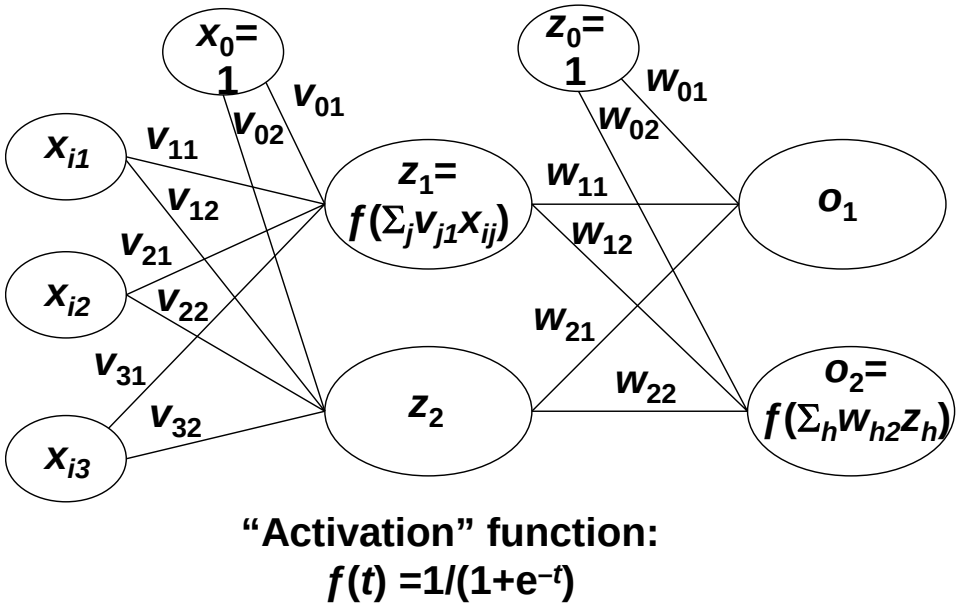


Figure 1. A sample neural network.

T is computed as the summation of input node-arc weight products into each hidden node. Last, the first step is repeated over the next set of nodes and arcs. That is, the hidden node-output arc products are summed. The simplest neural network is one having just two layers, input and output nodes, where the value of an observation is the activation function applied to each input node-arc product. This is just a simple logistic regression.

The three resulting models were used to determine if the columns contained data for worthwhile analysis, apart from the budgeted amounts. The logistic regression performed almost as badly the naïve model, which would be to state that amount of every budget line is above the median. The neural network was able to predict with over 80% accuracy which program lines would fall above the mean. However, the regression tree actually outperformed the neural network, achieving the highest classification rate. The reason for this is that the individual budget amounts correspond highly to four distinct column variables which describe where the software maintenance is performed and the size of the program. These specific results are contained in Appendix D.

C. METHODOLOGY

All figures were converted into base year 2005 dollars. Inflation indices were obtained from the FY2008 Naval Center for Cost Analysis Inflation Calculator. The Other Procurement, Navy (OPN) appropriation element was used. Parametric and non-parametric techniques were used to analyze the data. Student t-tests were performed to compare the total annual amounts spent on software support. A separate calculation was performed to compare the rate of increase in spending from year to year. The t-values were calculated and adjusted for sample size and variance.

The fiscal year of request, descriptive name (coded as EINOMEN), unique program identifier (PRON), amount funded, and total amount requested were extracted from the complete data set. Microsoft Visual Basic for Applications (VBA) procedures were written in order to sort the data, extract duplicates, and enable some non-parametric analysis to be performed. For most support programs in the TACOM data set, there is a stark difference between the amount funded and the amount requested. In most years, the data set reflects that most programs received either all or none of the requested funds. We suspect that the reasons for this are:

- The collection of programs are at different development states. Consequently, many have other lines of funding.
- Some programs may be forecasted an amount for a given year, with the intent to invest those funds over a number of years. This is known as a "level of effort" support. It is acknowledged that most programs do not receive the necessary funds projected to provide full software support to a weapons system.

For these reasons, the amount funded is not a reliable metric for analysis, particularly in years that have not been executed yet. This amount is analyzed, but a higher fidelity is given to the amount of funds requested.

THIS PAGE INTENTIONALLY LEFT BLANK

III. ANALYSIS

A. T-TESTS

Studentized t tests were calculated to make two comparisons. T tests are utilized to test hypotheses about what the data represents. In this case, we wanted to test whether the annual amounts of funding had similar means and variances between years. If the metrics were similar, it would make sense to perform additional analysis, developing a model describing the changing budget amounts. If the metrics were proved to be not similar, the thesis would focus on describing the differences, and interpreting what the differences represented.

First, year-to-year comparisons were made of mean amounts requested, converted into Constant Year 2005 dollars (CY05$) using the Naval Center for Cost Analysis (NCCA) Inflation Calculator for FY08 Budget, Version 1, and the Other Procurement Navy (OPN) Index. The t test required an adjustment, since the earlier reported years appeared in fewer POMs of this data set, while middle and later years appeared in most POMs. The smallest year group, 2002, had just 33 programs reporting, while the largest number of programs was 72, reported in 2011. It was assumed that the variances were heteroscedastic (or unequal), a reasonable conservative assumption given differing sample sizes. Accordingly, the following adjustment was made:

$$t = \frac{\overline{X}_1 - \overline{X}_2}{s_{\overline{X}_1 - \overline{X}_2}} \text{ where } s_{\overline{X}_1 - \overline{X}_2} = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}$$

Figure 2. Adjusted t for varying sample size and variance

Second, year-to-year comparisons were made on the rate of increase in amounts requested. Results from both sets of t-tests are contained in Appendix E. The data

suggests that there is not enough to reject the hypothesis that the mean, variance, and rate of increase are the same from year to year. Accordingly, the next phase of the analysis focused on modeling the annual amounts.

B. ANNUAL AMOUNTS

Analysis was performed to attempt to fit a model to the annual amounts budgets. If the annual increase followed a predictable form, this information would be useful to planners and professionals. A plot of the totals and a simple linear regression show the results in Figure 3.

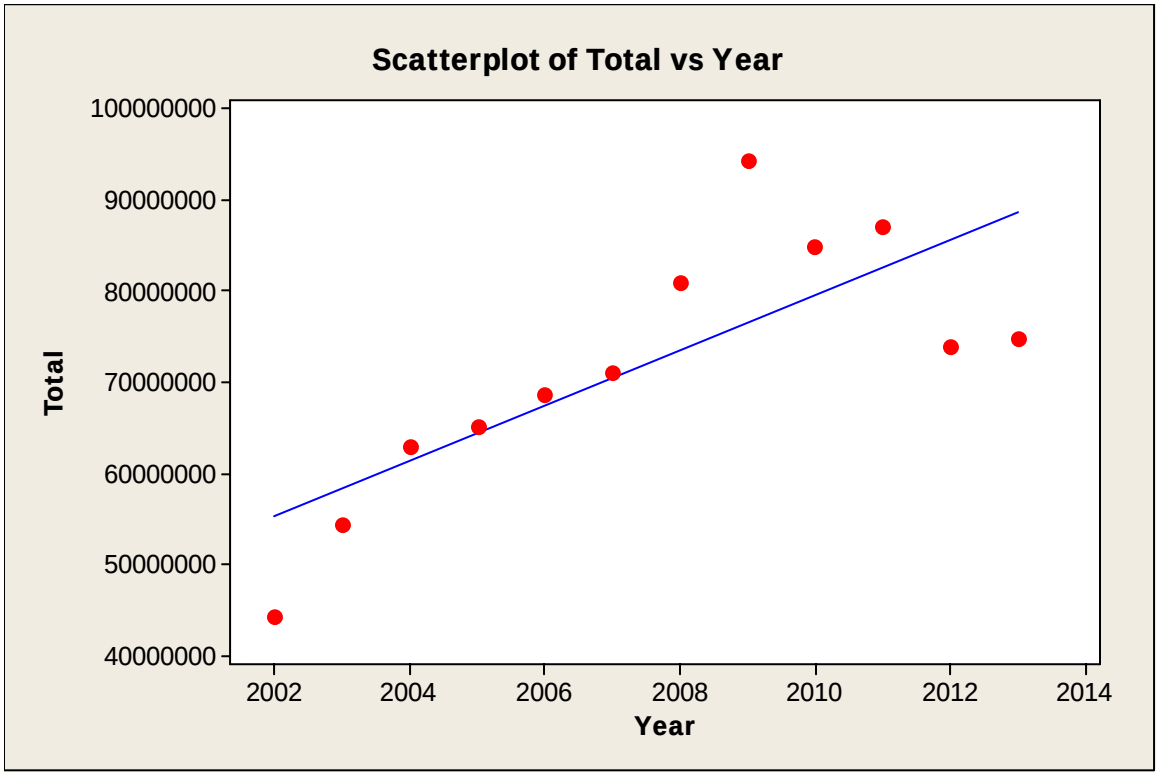


Figure 3. Annual Totals of TACOM Weapon Systems.

The regression fit takes the following form with the resultant r-squared:

$$y = 3{,}040{,}000x + 55{,}200{,}000$$

$$R^2 = 0.596$$

The fit simply states that the base year is budgeted for $55.2 million CY05, with an annual increase of just over $3 million. A good r-squared value does not necessarily imply a good model (the model may simply over fit the data); however, a poor r-squared is sufficient evidence of a model that does not fit the data well.

A simple second order polynomial provides a better fit and r-squared value, but there are reasons to assume that the yearly amount is increasing at a constant rate. The total amounts appear to ramp up in the early years and decline in the last three years of the data set. Seventy or more programs are represented in years 2009-2012, while only 54 and 56 are reported in the last two years of the data set. Likewise, the years 2002-2004 only have 33, 45, and 44 distinct programs reporting respectively. This is an artifact of the data set. The POMs are from fiscal year 2003 through 2008. Year groups in the middle are simply contained in more of these individual POMs, while early and end years are not.

In order to create a better model predicting annual total amounts, an adjustment must be made to account for the varying sizes. The data does not exist to make an accurate adjustment for this difference in sample size in different years. Accordingly, an adjusted total was derived from the data, using reasonable assumptions. Various methods were attempted. Adding the median for each perceived missing value had no effect on the resulting linear regression. Adding the mean resulted in a severe distortion. The totals between the year 2002 and the year 2011 are not that different. It appears that both years have many of the same expensive efforts budgeted. The means, however, are quite different, since 2011 has over twice the amount of reporting programs. The most tractable model takes the following form:

$$y = \left(n_{max} - n_x\right) X_{.625} + X$$

Where n_{max} is the maximum number of reported programs in all years, n_x is the number of reported programs in the given year, $X_{.625}$ is the 62.5th percentile of the given year, and X is the previously calculated total amount for the year. The resulting plot of adjusted annual totals is depicted in Figure 4.

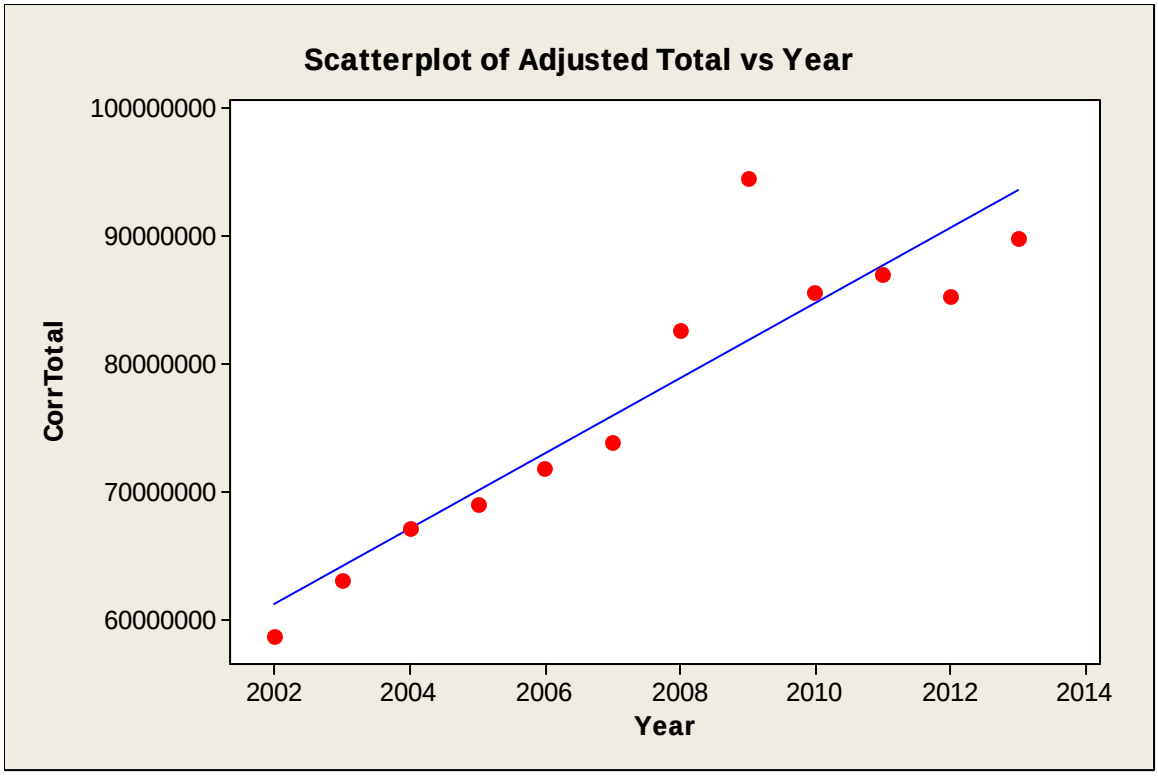


Figure 4. Adjusted Annual Totals of TACOM Weapon Systems.

The result is a reasonable approximation of the form the data would take, if all reported data from the early and late years was available. The linear regression is essentially the same, but with better r-squared.

$$y = 2{,}950{,}000x + 58{,}300{,}000$$

$$R^2 = 0.840$$

The choice of the 62.5th percentile is an admittedly arbitrary selection, but was made with consideration of the distribution of data. The 62.5th percentile represents a reasonable compromise where the adjustment performs some significance, yet does not

perform a calculation out of scale in different years. While the 50th percentile (the median) for this parameter had no effect, larger values such as the 75th percentile completely distorted the annual totals. The reason for this is that the data follows a different distribution for different years. For some years, the 75th percentile falls about the mean, while for others it does not. Table 1 depicts annual totals, means, and the 50th, 62.5th, and 75th percentiles. Shaded cells correspond to values above the mean.

	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
Total	44,221	54,469	124,211	123,488	68,707	71,001	81,047	94,439	84,995	87,051	73,998	74,922
Mean	1,340	1,210	2,700	2,205	1,165	1,224	1,266	1,330	1,214	1,209	1,370	1,338
X.5	112	102	103	118	115	150	151	163	188	215	233	264
X.625	370	319	257	233	209	210	211	235	336	357	634	932
X.75	1,119	1,089	1,501	1,100	877	953	1,649	1,415	992	1,884	2,087	1,996

Table 1. Budgeted Amounts in Thousands CY05$.

In the first six years of the data set, the 75th percentile falls below the mean, but for five of the last six years the 75th percentile is above the mean. The reason for this change is that the median changes dramatically between years. Between 2002 and 2013, the annual budgeted median doubles, as depicted in Figure 5.

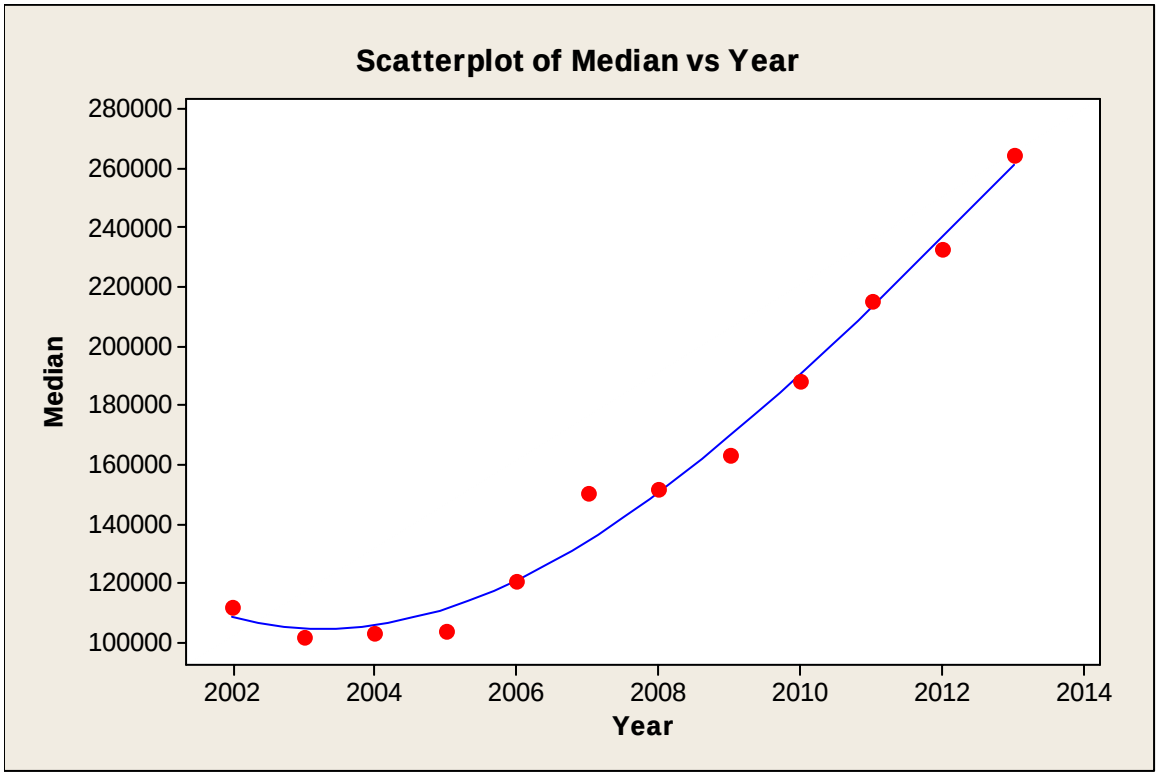


Figure 5. Annual Medians of TACOM Weapon Systems.

The expected median value is well fit for the database using a third degree polynomial. However, there is no fundamental reason to believe that a third degree polynomial underlies the budgeting problem.

$$y = -62.058x^3 + 375120x^2 - 8E+08x + 5E+11$$

$$R^2 = 0.9897$$

As was discussed earlier, the suspiciously high r-squared suggests that the plotted medians is over fit by the polynomial function. However, this distribution is reasonably well described with other functions, such as the exponential, which also fits the distribution better than a linear model. The implications of the median are discussed in Chapter IV.

C. DATA DISTRIBUTION

Developing a distribution that describes a dataset's characteristics can be a useful tool in determining what the data represents. After analyzing different functions for individual year groups, it was determined that the gamma distribution was the most informed choice of model. The gamma distribution is typically used to model observations such as rainfall frequency [10], mean time to failure, and other lifetime or survival distributions. The shape parameter affects the steepness and general form of the distribution. For small values of the shape parameter, the distribution is extremely skewed. For large values, the distribution becomes approximately normal. The scale parameter impacts the spread of the distribution. Small values will result in a model representing data that is close together. For larger values, the distribution is more spread out. The 2-parameter gamma distribution has expected value and variance of the following form:

$$\mathrm{E}\left(\mathrm{X}\right) = \rho\beta \qquad\qquad \mathrm{V}\left(\mathrm{X}\right) = \rho\beta^2$$

In the equations ρ is the shape parameter and β corresponds to scale. With threshold as a third parameter, the expected value simply adds the threshold value to the $\rho\beta$ product. This allows for a better fit of the data. Threshold has no effect on the variance.

Figure 6 depicts one of the best fits for a 3-parameter gamma to the empirical cumulative distribution function (CDF) of a particular year, the year 2013.

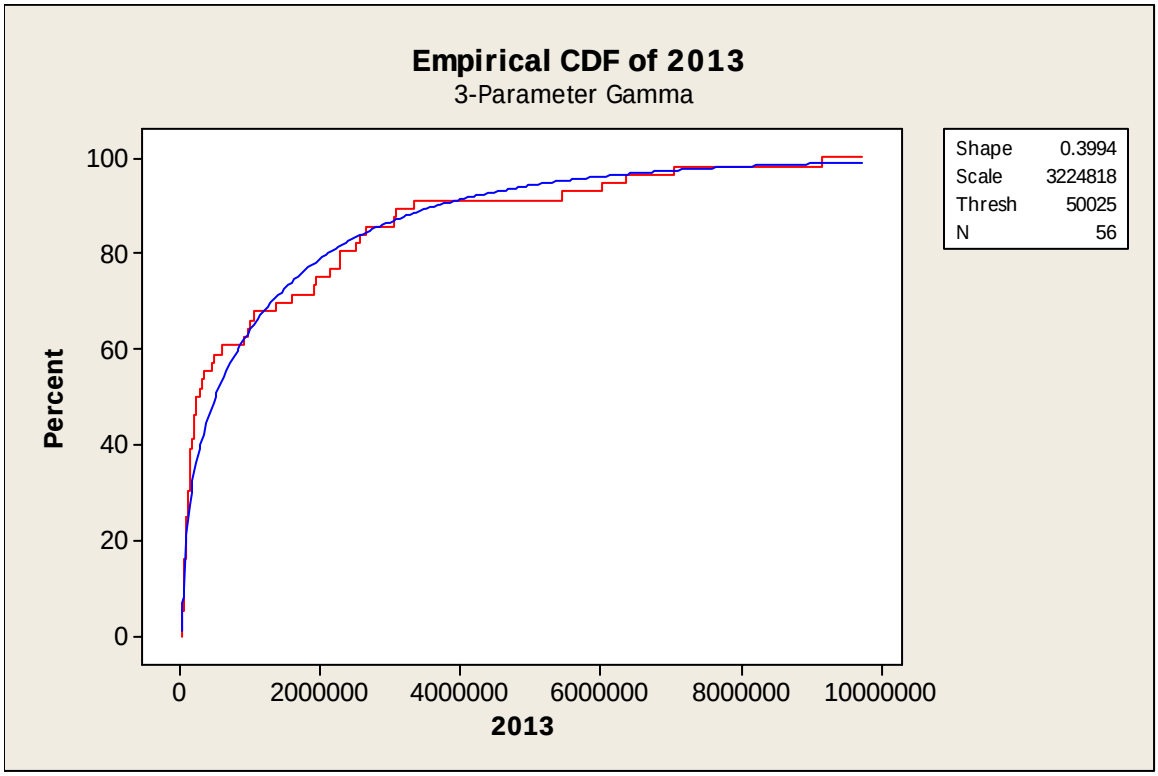


Figure 6. 3-parameter Gamma distribution vs. Empirical CDF.

The plotted track appears to follow the empirical distribution fairly well. There are numerical methods to determine goodness of fit, but for this instance it is more easily interpreted by analyzing other attributes of the data set. One such metric is the probability plot.

Figure 7 depicts the probability fit of the same gamma function to the dataset represented in 2013. The distribution parameters, found in the legend, remain the same as in Figure 6.

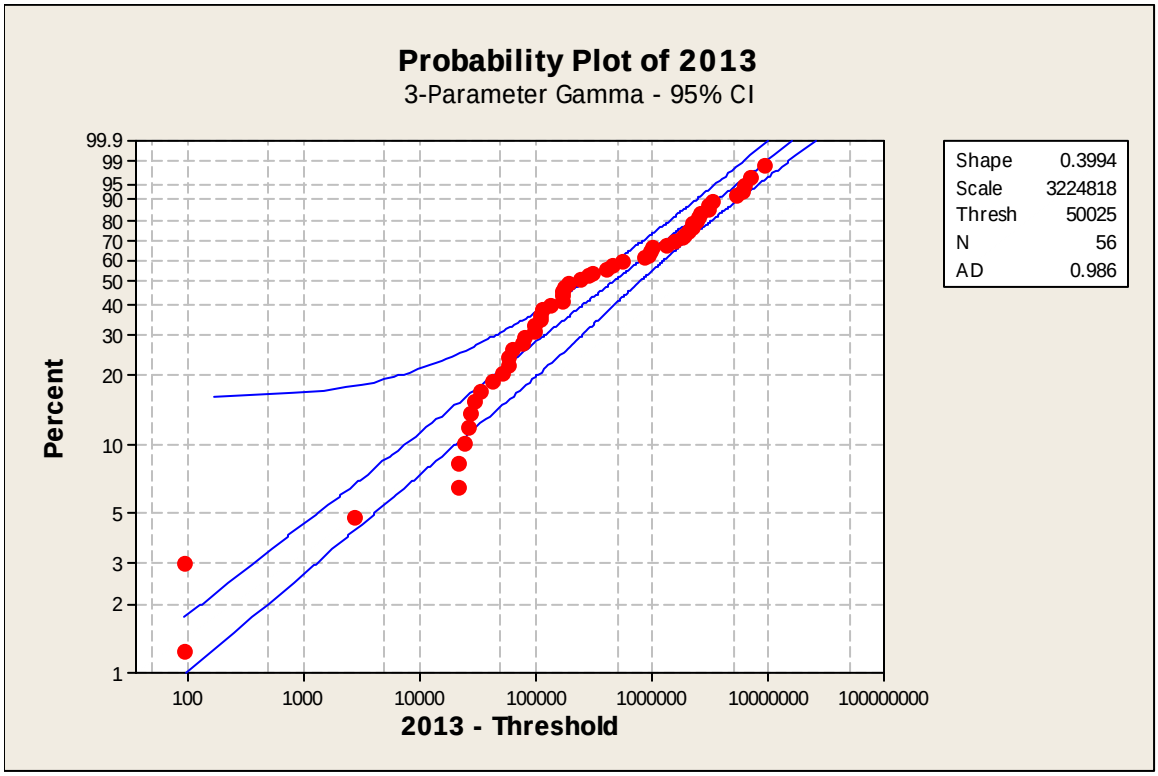


Figure 7. Small values outside the 95% CI, but large values are well modeled by the gamma with these values.

The probability plot highlights three key features of the data set that are not readily apparent from the empirical CDF plot. First, a large percentage of the data values are small and the gamma distribution overestimates the amount. This is indicated by the number of red dots falling below the lower 95% CI band. Second, while it appears that the model remains close for middling values, the model performs badly for this portion of the data. This is indicated by the value approaching and breaching the upper 95% CI band near the 50th percentile. Third, the data set does not appear to be normal or approximately normal. This fact is apparent by the slight 'S' shape that the distribution of red dots takes.

These indicators of what form the data takes are reinforced by another view of the data, a simple histogram in Figure 8.

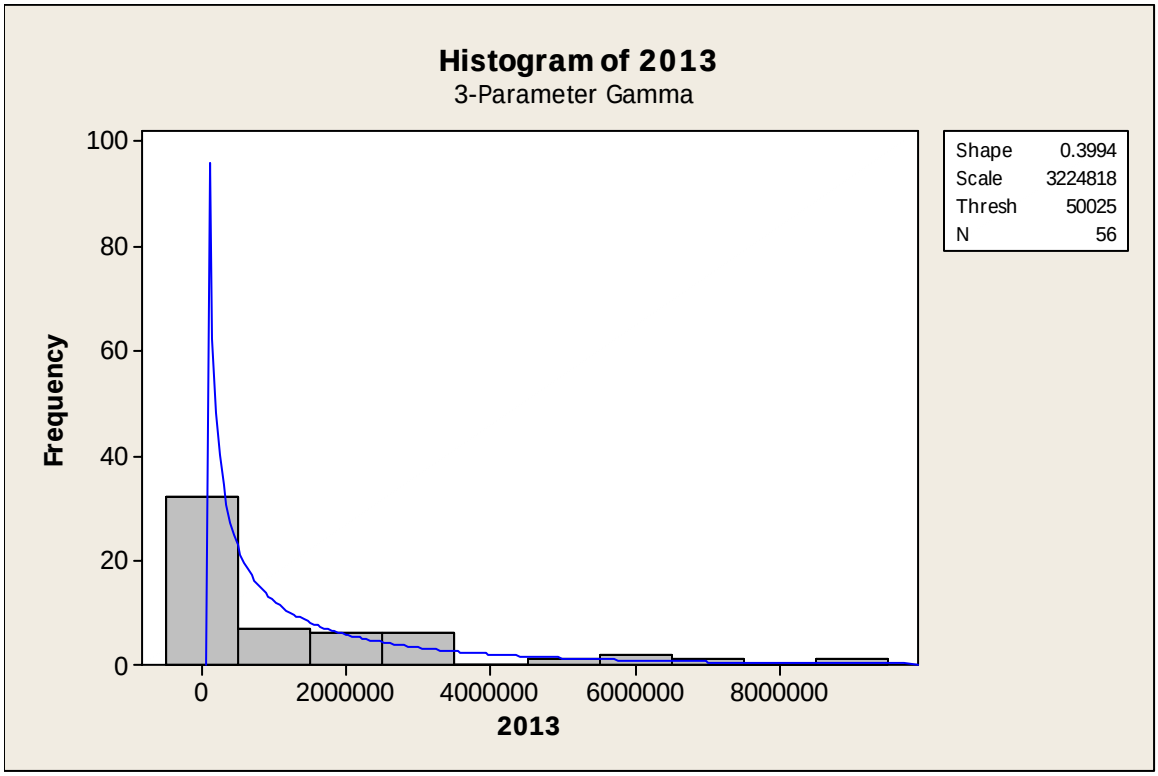


Figure 8. The distribution is not well fit for small values, but the gamma approximates the long right tail well.

The high left-hand peak overestimates what the histogram displays. This reaffirms what the probability plot suggested, that the selected model is inaccurate for predicting the number of small values. The right tail of the distribution highlights the source of the problem. The model has made a computational trade off in the heavy left tail, so that it will fit the long right tail. In order to discover a distribution that provided a better fit, the data was transformed with the logarithmic function.

Data transformations are typically performed for one of three reasons: failures of normality, linearity, or homoscedasticity. Also, data transformations are usually needed to prepare for regression analysis. For regression analysis to be valid, the data must follow a normal distribution (expected mean and variance), be linear (a linear function can be applied to the data to predict an outcome), and homoscedastic (data in the sample have equal variances). However, we are not comparing the budgeted amounts in the data set to a predicted amount. In this case, the transformation was performed because the

data is out of scale with itself. In each individual year, there are many small values, and the spread of values is orders of magnitude apart. Figure 9 shows the plot of the transformed data, with the resulting gamma model and empirical CDF.

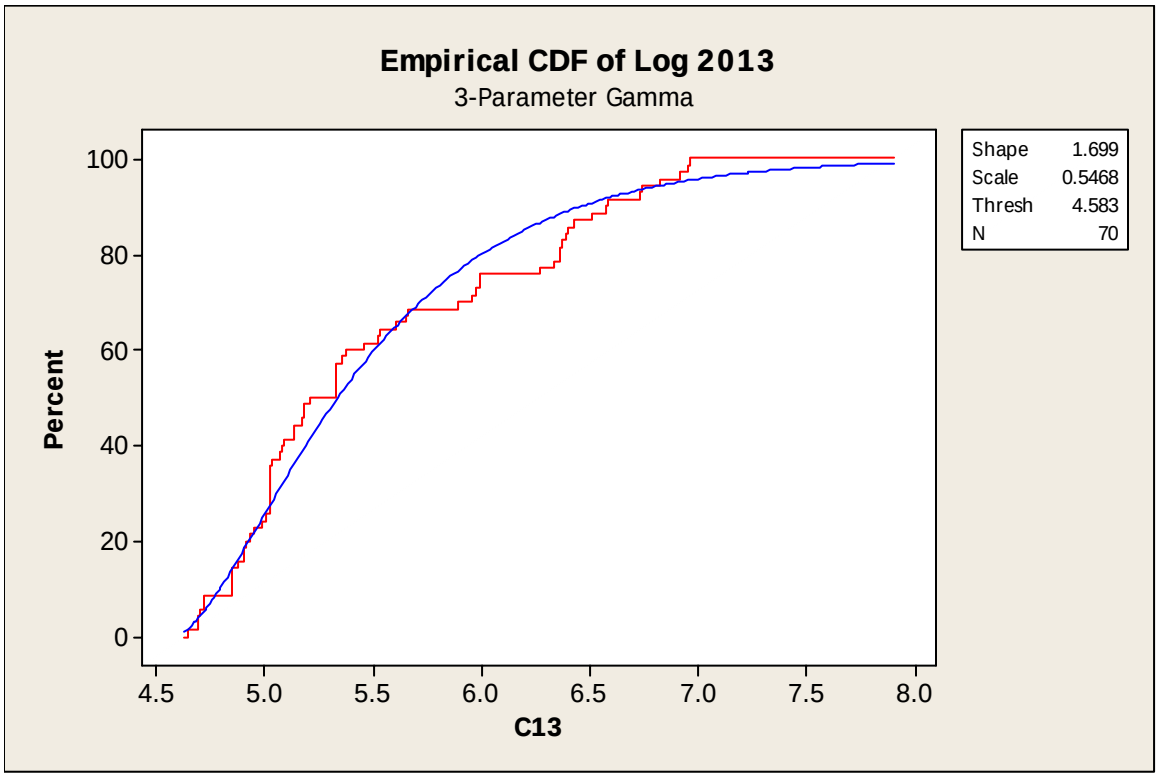


Figure 9. Small values are better fit by the transformation.

The prediction parameters in the legend have changed, which should be expected. The graph is not as aesthetically pleasing as the empirical CDF of the unaltered data. There appear to be larger differences between the given data and predicted distribution. However, analysis of the probability plot and histogram reveal this to be a better informed model. Figure 10 depicts the probability plot.

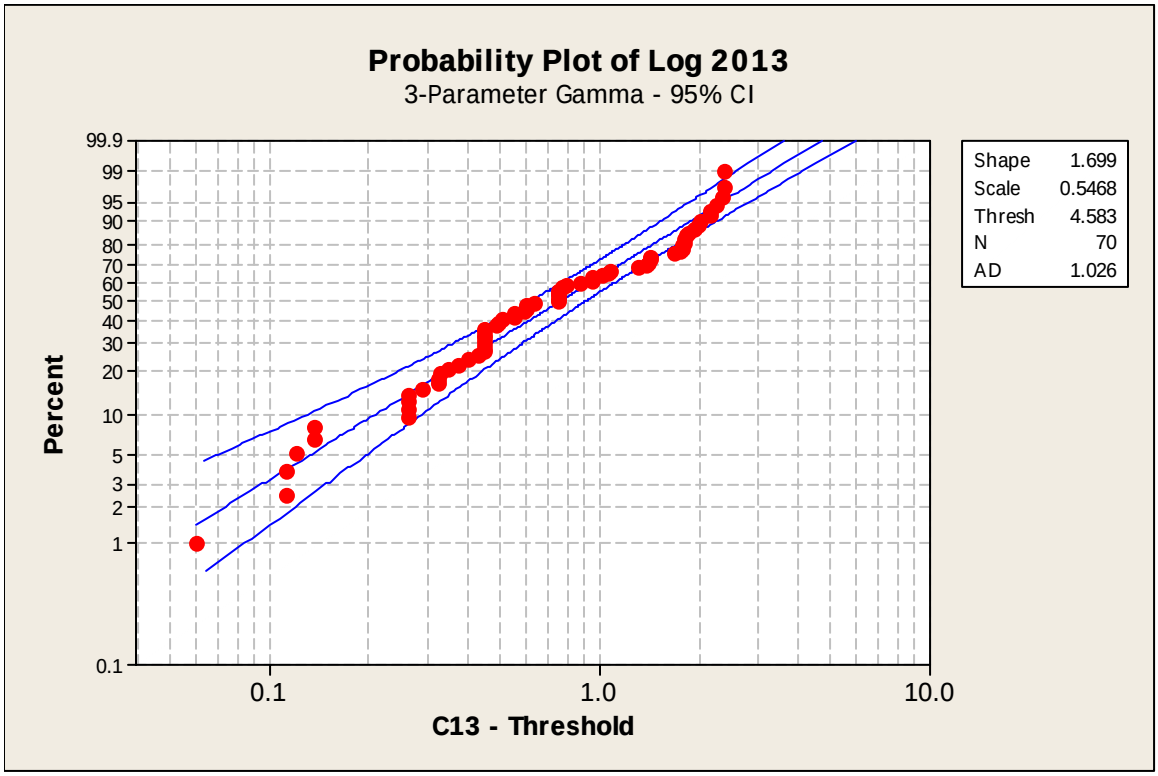


Figure 10. All small values within the 95% confidence bands.

Virtually all of the observed data points fall within the 95% CI. The model performs considerably better for smaller values. In the untransformed histogram, the high peak for low values overestimated what the histogram displays. This reaffirms what the probability plot suggested, that the selected model is inaccurate for predicting the number of small values. While the gamma probability plot on the raw data performs well on large values at the expense of predicting small values well, the gamma on the transformed data has made a trade-off in the opposite direction. This is depicted in Figure 11.

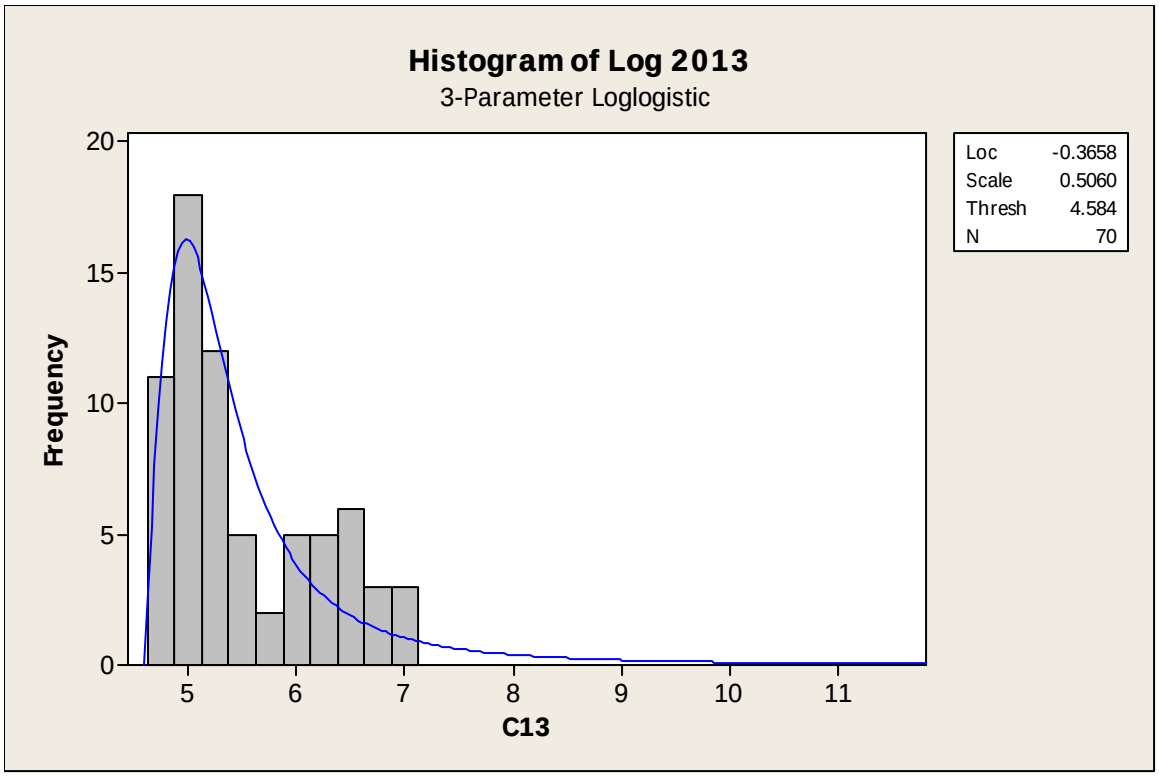


Figure 11. A better fit for small values.

It is possible to fit a distribution to each year represented in the TACOM database. However, after performing separate fits on a few years contained in the dataset, it was determined that this is not a valid analysis. The empirical distribution for 2013 was chosen because it was one of the best fits of the available data. While the 3-parameter gamma fits the year 2013 well, the distribution performed poorly on other years. It was observed that other distributions, such as the 3-parameter logistic and 3-parameter lognormal, provided better fits for some individual years. Results of attempts to find 3-parameter gamma fits for the individual years are contained in Appendix F. Fitting a distribution for one year did not yield any useful information in attempting to fit a successive year. This analysis yielded no suggestions as to what is the distribution of data in years outside the data set, other than it would most likely be best fit by one of the many models used. Fitting a distribution for the entire data set performed more poorly than any of the individual years. The reason for this lack of fit is fairly interpretable.

Since the medians are different for individual years, it is expected that the distributions are indeed different. We shouldn't expect to find a better distribution to the entire data set, since each individual year has a heavy left tail. Combining the data only exacerbates this fact, as show in Figure 12.

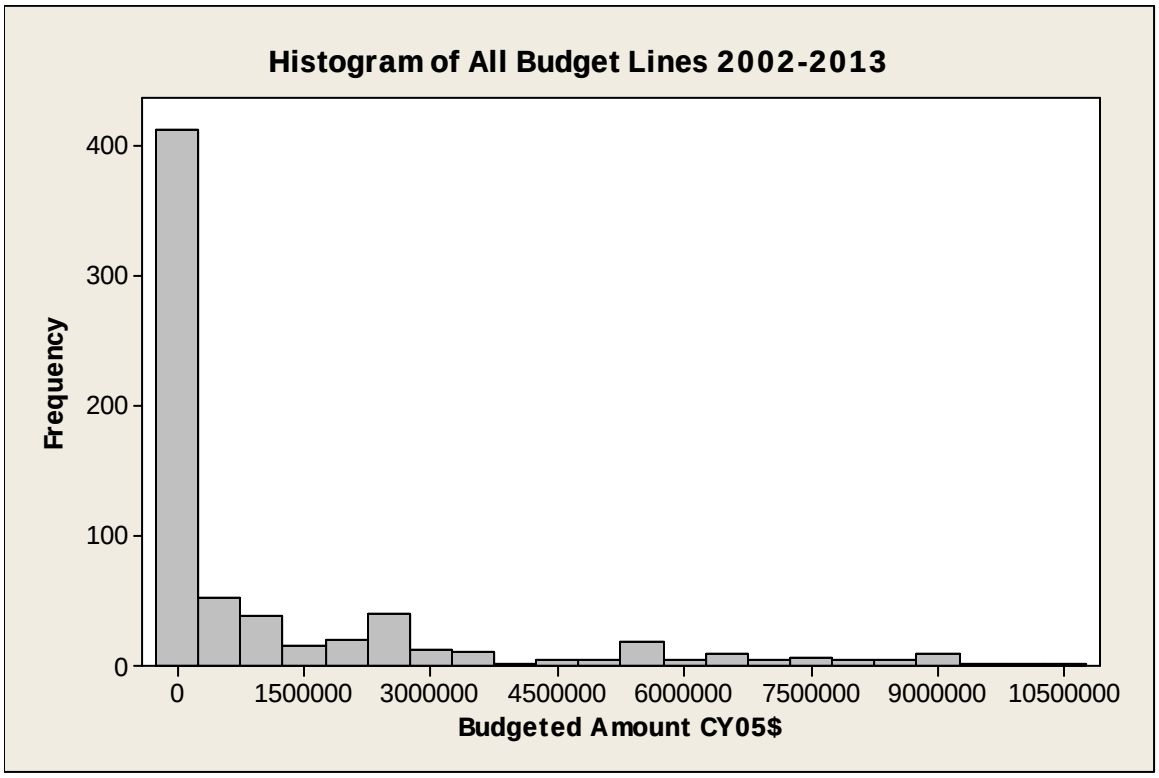


Figure 12. Histogram of all budget lines in years 2002-2013.

D. WEAPON SYSTEM DATA

As previously displayed in Figure 5, the medians are different between years and are changing at a predictable rate. Fitting a distribution to the entire data set loses this sophistication. Most of the unique PRONs in the database map to a specific ground combat system, as described in the EINOMEN field. By combining these budget lines, we can chart yearly budgets for the associated weapon systems. The resulting annual budget by individual weapon system is depicted in Figure 13.

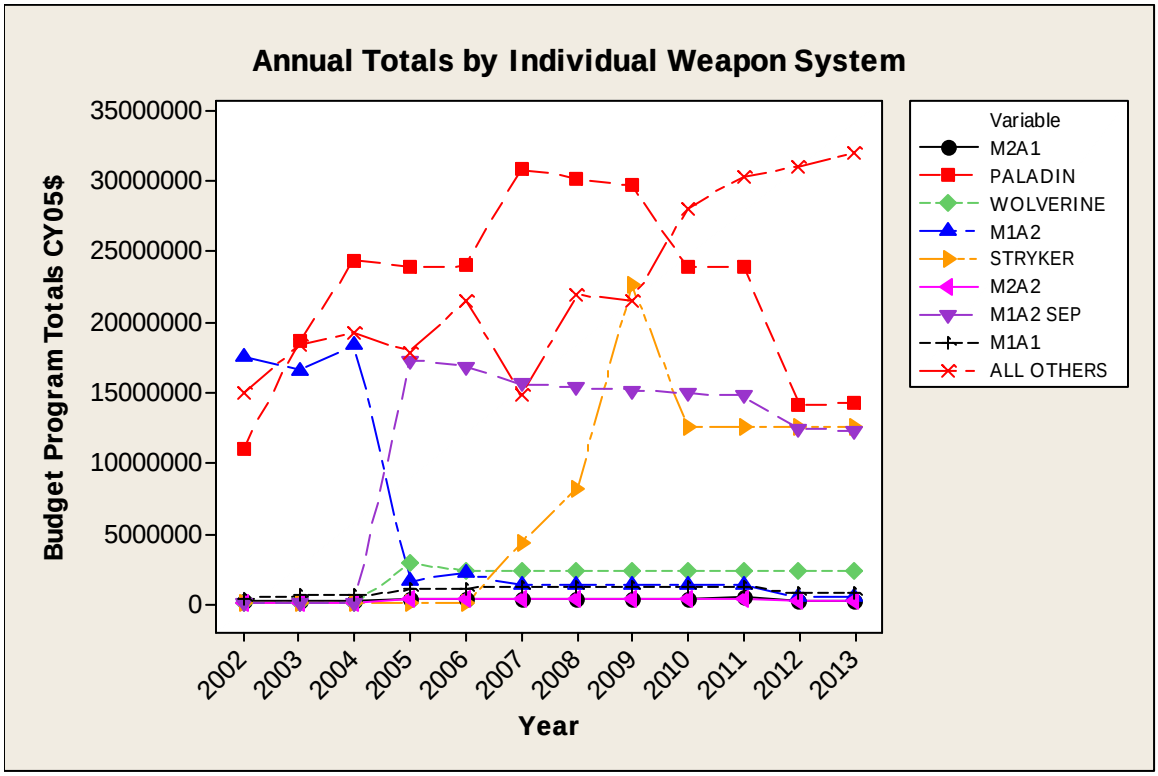


Figure 13. Annual Totals by Individual Weapon Systems.

The "ALL OTHERS" data field corresponds to a large share of mostly small budgeted systems that do not correspond to a specific platform. Largely, these are NBC systems, munitions programs, fire control systems not associated with specific platforms, threat warning devices, and joint effects modeling programs. As has been stated before, there is evidence suggesting that the annual amounts allocated to each weapon system are dependent on each other. However, the plot in Figure 13 does little to suggest a dependence relationship between any of the weapons systems. Many of the programs, M1A1, M2A1, M2A2, Wolverine, are essentially flat. This reflects a "level of effort" budgeting process. This means that weapon systems are given a predetermined level of funding, without regard of the estimated work to be accomplished or other predictive measures. Otherwise, there are not many comparisons that can be made within the data in the way it is presented. However, a few inferences can be made, based on conversations with SMEs.

It was learned that the M1A2 and M1A2 SEP programs are essentially the same system. Many of those budgeted programs simply changed names in 2005. Accordingly the programs have been combined. The Paladin is the most costly single system in the data set and it leads by a large margin. After talking with several software maintainers, it is their opinion that the Paladin is the highest priority software system. The suspicion is that the Paladin receives the budget it requests, and that other systems have their budgets reduced as a consequence. Accordingly, the Paladin has been combined with the "ALL OTHERS" data line. The result from these two changes is depicted in Figure 14.

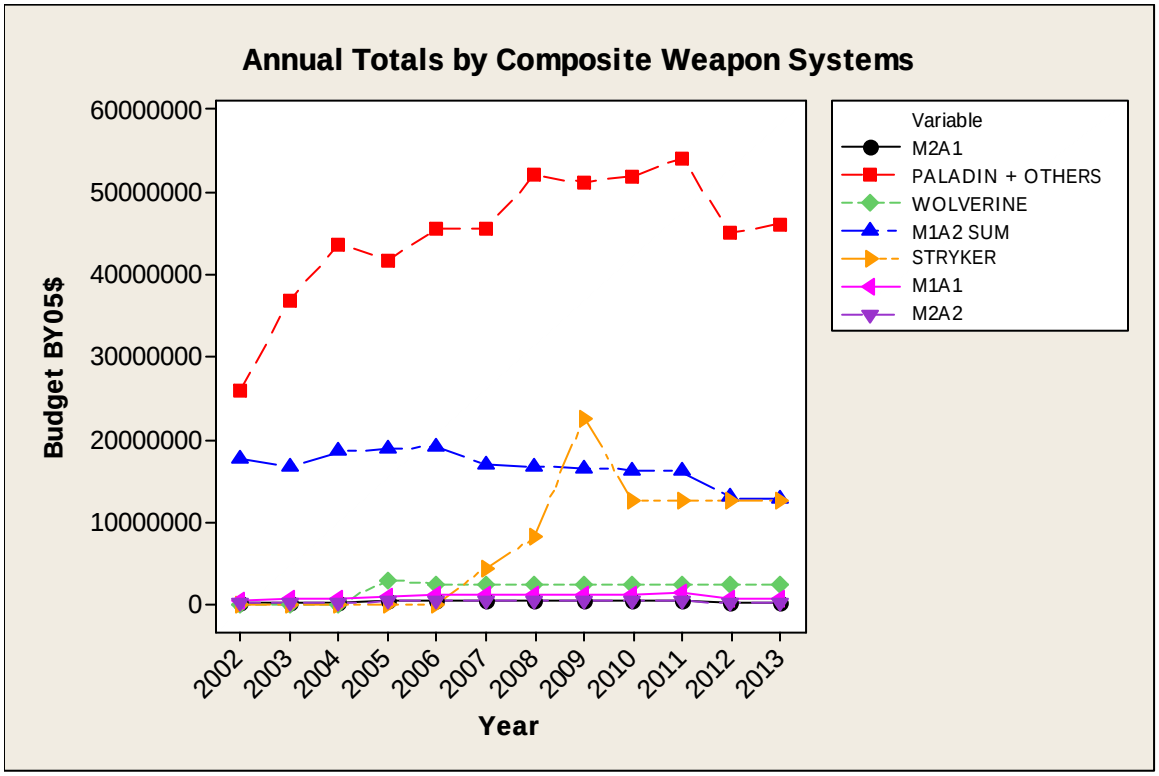


Figure 14. Annual Totals by Composite Weapon Systems.

It is important to reinforce that the data necessary to support these assumptions is not contained in this thesis. These inferences were made based on reasonable assumptions and conversations with SED professionals. The resulting picture creates strong suggestions about what the data represents. The M1A2SUM is relatively flat. Stryker budgeting, which doesn't appear until 2007, appears to also flatten in the out-

years. It appears that by 2012, it has been determined that the Stryker and M1A2 programs will be funded at the same amount. Lastly, the combined Paladin/All Others summation follows a much more predictable pattern than either individual data plot in Figure 13. This suggests that there is in fact a relationship between the two. The suspicion is that Paladin, being the top priority, receives the desired amount of funding, while these other smaller systems suffer the fallout. For many reasons, there is not enough evidence to suggest this is a definitive conclusion. The budgets for the later years in the database, where Paladin funding decreases and All Others increase, have not been finalized. The 155mm Crusader, which was designed to replace the Paladin, was canceled in 2002. It is possible that the planned decreases in Paladin funding were intended to be sourced to the Crusader. It is difficult to make valid suggestions about what the data represents for individual weapon systems until the budgeted years are finalized and executed.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY OF FINDINGS

The first observation from our analysis of the data is that there is no difference, on a year-to-year basis adjusted for inflation, of the total resources budgeted for software maintenance for ground combat vehicles. This means that there appears to be a set budget for software maintenance for ground combat vehicles.

The second observation from our analysis of the data is that when the annual amounts spent for software maintenance for ground combat vehicles are adjusted for inflation by converting them to constant year dollars, these total amounts follow a simple linear regression, and the level of effort provided for overall program growth is growing at a constant rate. As expressed in the data, this is estimated growth is $2.95 million dollars a year, or a 5.06% rate of growth.

The third observation contrasts with our first that there is no difference of the total resources budgeted for software maintenance for ground combat vehicles. That is, within a given year, there are stark differences in the amounts that different programs are allocated. It appeared that, on an annual basis, the budgeting offices have different priorities for the different weapon systems, and these priorities show up strongly in the amounts allocated to each. This is evident from two metrics which describe the data. First, it's clear that specific software intensive systems require much more funding, as shown in Figure 15.

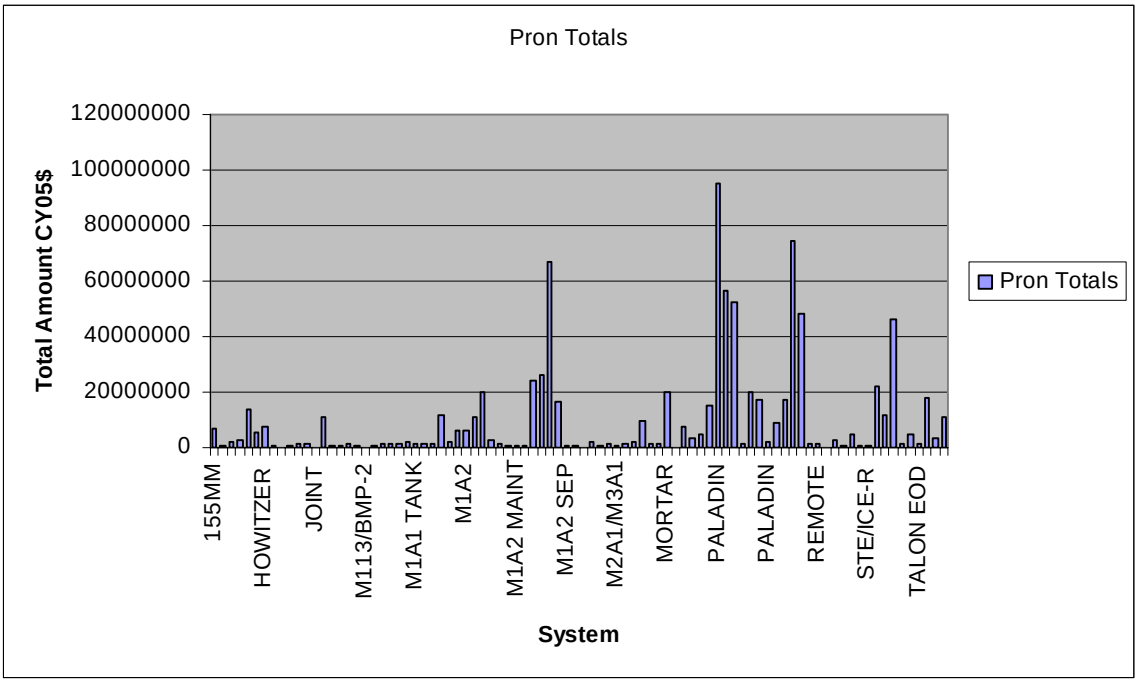


Figure 15. Annual Totals by Individual Weapon Sub-programs.

The Paladin Self Propelled Howitzer, with its complex fire control system, has many sub-programs more expensive than the entire M1A1 software support. This result is to be expected and, with more data about each program's development, could be modeled in a significant way to inform decision makers of the expected support costs.

Second, the most expensive programs are receiving a larger and larger share or the budget. This result is best described by the increasing annual median. The median appears to increase at a polynomial rate and more than doubles between the years 2002 and 2012.

From the observations above, we concluded that the amounts allocated to different programs within years are not independent. If the total budget follows a fixed growth rate and there exists different priorities among the weapon systems, then fortifying one system cannot be performed without lessening the support for another.

B. SPECIFIC RECOMMENDATIONS

There are many methods to reduce the life cycle costs of software intensive systems. Software design should consider software maintenance in mind, since that is rapidly becoming where the lion's share of the costs are incurred. It is recommended that software support facilities be encouraged and funded to improve their processes. Studies have shown that software engineering organizations with higher CMM ratings have a smaller percentage of code that needs to be reworked, receive fewer defective reports from consumers, and have a higher overall productivity while writing code [11]. Research from software engineering centers and some rudimentary analysis of the database provide many areas to improve software efficiency. In 2001, General Dynamics conducted an internal study in which quality, rework, and productivity performance were measured against CMM. The results are detailed in Table 2.

CMM Level	Percent Rework	Phase Containment Effectiveness	CRUD Density per KSLOC	Productivity (X Factor Relative)
2	23.2%	25.5%	3.20	1 x
3	14.3%	41.5%	0.90	2 x
4	9.5%	62.3%	0.22	1.9 x
5	6.8%	87.3%	0.19	2.9 x

Table 2. Performance vs. CMM Level.

Both CMM level and the other benchmarks were determined by internal metrics. In this study, the process improvements cannot be solely attributed to improving the CMM level. Many of the organizations in the study had put in place initiatives beyond the recommended CMM levels. SEI created a software initiative in late 2006 called Improving Processes in Small Settings (IPSS). The leading research in the area suggests that process improvement should be an ongoing, as needed activity for software developers of any size. A study profiling Infotech, a level 5 CMM developer, shows how the company was able to track productivity and size variation as a project team grew from 11 to 45 developers over an 11 month period [12]. The study concluded that the traceability Infotech provided served as valuable documentation for the maintenance phase.

An example of where this is relevant to SEDs and the programs listed in the TACOM database is in regression testing. For many software professionals, regression testing is an automated process by definition. However, at TARDEC's Next Generation Software Lab, such testing is still being performed by hand. The specification to test a new version of code that runs on one of their weapon systems runs a few thousand pages.

Most software is developed under a cost plus contract. As of this writing, this is true of most if not all of the FCS software contract awards referenced in Appendix B. The vast majority of this software has not been completed yet. In large part the contracts were awarded based on the contractors past performance, software design, and other credentials. A simple solution to reduce development costs is to allow contractors to prototype and present their own solutions at a fixed price. The government could then choose to buy or not to buy the software, depending on the interoperability, maintainability, and current relevance of the system. This allows the government to procure more competitive bids for programs throughout the software life cycle.

There are various lessons that are not supportable by the level of detail contained in the database, but have been learned by the subject matter experts (SMEs) performing PPSS on weapon systems today.

- Constant improvement to all software processes should be made a priority by program offices. Studies have shown that appropriate constant focus

on improving internal processes has a positive effect on software quality, productivity, and reduces rework. Software intensive programs should invest in individuals trained in improving the software process and retain this knowledge for the life of the program. Some SEDs have already trained and certified staff members in SEI's Personal Software Process/Team Software Process. But in speaking to the smaller software maintainers, they do not have this level of support, and process improvement is not seen as a priority by higher management.

- It is recommended that software support budgets be maintained throughout the year. This would be desirable of any program of any type, but it is more crucial in the case of software because the cost of interruption is difficult to quantify. This is a problem that is not well encapsulated in the TACOM database since its figures appear in yearly increments. However, from conversations with software professionals at the SEDs, this problem has known consequences. Software maintainers know empirically what support can be provided, even given limited funding. This is not a simple or linear process. Given an amount of funding, managers make decisions on permanent hires, contracting work, quality assurance, and support personnel both in the home office and providing updates to warfighters. Withdrawing 25% of the budget usually has a consequence larger than 25% of the expected effort. Software facilities are often a convenient entity to stash or hide funds for later withdrawal for use on other program segments. It is recommended that software budgets be maintained and not changed, in order to allow software engineers to better plan their processes and achieve more efficient use of budgeting dollars.

- Recommend that the funding of software support become programmed into the life cycle of the weapon system as a separate entity from regular maintenance. From a software professional's perspective, there is no clear cut difference between software development and maintenance. Software maintainers perform the same essential function as developers in the respect that they release new versions of the code, typically on an 18-month cycle. Yet custody of this critical technology is passed between entities with little transparency. From a practical standpoint, these divisions derive from the reality that money comes from different sources. The manner in which the funding is provided currently, many systems are fully funded for software only until the hardware system is actually produced. According to the SEDs, the software maintainers must wait until the system is a year out of production in order to receive Operation and Maintenance (OMA) dollars. There are many program offices for which software is a critical technology, necessary for the program's success. To maintain continuity in the software effort, program offices should have a civilian manager ensure that software decisions are fully embedded in LCC considerations. This position would also facilitate

rigorous software design and implementation [13]. An additional duty for this office is to preserve software data. Only by placing such a person in the program office is it possible to answer key research questions about the software's design and portability decisions, and what funding went into development, post development, maintenance, and improving process improvements.

C. RECOMMENDATIONS FOR FUTURE RESEARCH

Better data collection needs to be performed in order to make stronger inferences about the consequences of decisions in the life cycle of software systems. This is largely outside of the scope of an individual researcher. The Army's OSMIS website has been promising software data, but this effort remains under development. Even when the data has been completed, it has been indicated that the relevant metrics (SLOC, FP, productivity numbers, languages written) for building a data model will not be included in the near term [14]. Navy VAMOSC has some software data, but it is not easily retrievable, does not have a uniform breakdown, and does not contain metrics such as SLOC, function points, or project duration which are useful for formulating models. Valuable research could be performed analyzing the different software data methods between Armed Services, the relevance of metrics used to track progress and justify funding, and analyzing the effort it would require to create a Joint database on software LCC.

It is recommended that a study be conducted to analyze the trade offs from decisions made throughout a software program. The Aegis Combat System and Air Force F-22 Raptor programs are both desirable candidates for analysis, perhaps contrasted with each other. A compelling study could be performed comparing the costs of the Stryker ATGM (with its embedded training system) against the Paladin, with its associated costs for training, simulators, and operating and support.

Lastly, another study of interest would be to analyze and contrast the performance of the military and contractors as lead systems integrators (LSI) for large software programs. In recent years contractors have taken on the role of LSI. Boeing is the LSI for FCS. The Northrop Grumman/Lockheed Martin partnership failed on the Coast

Guard Deep Water program, as did General Dynamics in producing the Marines' Expeditionary Fighting Vehicle. The use of contractors as LSIs is an interesting development because they all involve high dollar programs incorporating cutting edge technology. It would be of interest to analyze the maturity of software under these failed programs, and what role, if any, software problems played in the failures.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A

Historical Examples of Software Problems

The city of Denver commissioned a new airport to be opened on Halloween 1993. One of the highlightshallmarks of the new airport was to be an highly automated baggage system. BAE Automated Systems was awarded the $186 million contract to build the baggage handling system and code the software. The system planned for 56 barcode scanners, 400 radio receivers, 100 networked computers, 4,000 independently operating carts, and 5,000 electronic sensors to implement the timely arrival of baggage. The type and scope of these components is similar tonot unlike many distributed and networked systems currently being procured by DoD program offices. In Denver, the baggage system was foreseen as the primary cost saver, allowing airlines to better plan their flight changes. However, chiefly because of the baggage system, the airport opening was delayed 18 months until May 1995 at a cost of over $1 million a day [15]. By the time it opened, costs in delays had exceeded the original estimate for the entire airport and t. The baggage system was finally abandoned in August, 2005.

Perhaps the most famous software failure was the first flight of the European Ariane 5 expendable launch system in June 1996. The Ariane 5 had not been fully flight tested because of high confidence in modeling and simulation of flights. As a result, the first flight carried $500 million worth of payload in satellites. Shortly after liftoff a software error occurred. A program segment using a decimal measurement attempted to convert the floating point number to a signed 16 bit integer. However, the input value was out range of the software andi.e. because of the number of digits, it could not be represented as a 16 bit integer. The run time error occurred in the active and backup computers, which both shut themselves down. The Ariane 5 lost all attitude control, started an uncommanded turn, and aerodynamic forces broke the vehicle apart. A separate onboard monitor detected the breakup and ignited explosive charges to destroy the vehicle in air about 40 seconds after lift off [16]. The code that caused the error was reused from an earlier vehicle where the measurement couldn’t become large enough to cause this failure.

Great software successes are not as climatic as their failures. In the early 1990's, the on-board shuttle group became one of the first four developers worldwide to attain the highest Capability Maturity Model (CMM) rating, Level 5. This group writes the code for NASA's space shuttle and is a branch of Lockheed Martin. Of 11 versions of the code that controlled the space shuttle during this period, just 17 total errors were detected in the code, which amounts to 420,000 lines in each version. A commercial version of similar complexity would have had 5,000 errors [17].

This is not to suggest that certain programs are simply bad and that NASA should be the gold standard for software programs. The Lockheed group's annual budget at the time was $35 million for this one software program. The group had 260 people working in support of it. Documentation for the program ran 40,000 pages. When the code actually runs on the shuttle, it uses four identical computers each running the same code. Clearly DoD can't match NASA in terms of resources. But the shuttle's group excellence is a clear example of benefits reaped by improving the software process. To better understand how to improve the software process for, first we must examine how software support is currently performed.

APPENDIX B

Boeing functions as Lead System Integrator. To develop software for FCS, Boeing has awarded contracts to Lockheed Martin [18], Raytheon [19], Science Applications International Group (SAIC which along with Boeing serves as Lead Systems Integrator for FCS), General Dynamics Robotics Systems (with Auburn University) [20], General Dynamics C4 Systems [21], BAE Systems [22], Curtiss Wright ,[23], General Atomics [24] InstallShield, iRobot Corporation [25], Northrop Grumman, United Defense L.P., Boeing Mesa [26], Lockheed Martin Missiles and Fire Control [27]. Additional software support comes from existing open source software designs such as Modeling Architecture for Technology, Research, and Experimentation (MATREX). Boeing has also named contractors that are developing supporting partners that will provide software to run services such as simulations, training, sensors, and integration: Austin Information Systems, Computer Science Corp, Dynamics Research Corp, Honeywell Defense and Electronic Systems, Northrop Grumman Mission Systems (along with the company's divisions in Electronic Systems and Information Technology Defense Enterprise Solutions), Telcordia (an SAIC subsidiary), and Textron Systems [28]. Testing support is being provided by Communications-Electronics Research, Development, and Engineering Center [29], Fort Lewis Electronic Proving Ground. The Army's Program Execution Office (PEO) Simulation, Training, and Instrumentation will also contribute by reusing existing components to help create the SoSCOE [30]. This list is not exhaustive. Many contracts have yet to be awarded at this date. Particularly absent are developers to build unmanned sensors, a key component that gives FCS its battlefield edge and lethality.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C

The history behind the F-22's software development is instructive. It is a sizeable program, just entering full production. Tthe F-22 program is highly software dependent. Lastly, the F-22 displays numerous preventable problems caused by a system's software design. When the F-22 contract was awarded in 1991, software development written for DoD was mandated to be written in the programming language Ada, under MIL-STD-1815A [31] and later MIL-STD-2167A [32]. Ada is a high level language similar to C/C++ that came about specifically as the result of a DoD study to reduce the number of programming languages used in development of DoD software programs. The "Ada law" was first implemented in 1987 and later revalidated as part of the annual Defense Appropriations Acts for fiscal years 1991, 1992 and 1993. It simply stated that all DoD software must be written in the programming language Ada, where cost effective, unless exempted by an official designated by the Secretary of Defense. The standards were later repealed in 1998, yet 80-85 percent of the avionics onboard the F-22 is still written and maintained in Ada.

The F-22 has become well known for software failures. The first Raptor prototype crashed in April 2002, after the contract had already been awarded. The cause of the crash was determined to be pilot induced oscillations, which were attributed to a software error [33]. The program is still behind schedule, mostly do to it's sophisticated software driven avionics. Even after twelve years in development, in 2003 the software running the F-22 was crashing, on average, every three hours [34]. In late February 2007, a flight of eight F-22's were flying from Hickam Air Force Base to Okinawa, Japan. Due to a software bug, upon crossing the international date line all eight aircraft dumped all computer systems. They lost all navigation, communications, and some fuel systems. The pilots had no internal attitude reference. The flight of Raptors maintained visual contact with their tankers and returned to Hickam, where fortunately the weather was good. The Air Force employed "tiger teams" to debug the code and the software problem was fixed within 48 hours.

The full procurement price on the Raptor is estimated at $34 billion, with an additional $28 billion sunk on the aircraft's research, development, and testing. The planned buy is 183 aircraft, $339 million each based on total program costs. The incremental cost for a new Raptor is $120 million, but current required changes are expected to raise the cost to $166 million [35].

The Aegis Combat System is an interesting case study for many reasons. It is a long standing program, highly software intensive, has been installed on numerous platforms, and has been extremely successful in terms of operational effectiveness and suitability. In the 1960s the U.S. Navy began development on a program to defend its ships against anti-missile attacks. The program was called Advanced Surface Missile System, but was renamed Aegis in 1969. The Aegis is currently employed on 107 surface platforms predominately for the United States, but also for the Japanese, Norweigan, Spanish, and South Korean Navies. From a software perspective, the Navy program offices have recently made decisions over the Aegis's life cycle which contrast with those of Army and Air Force programs.

For many programs it is difficult to determine when development stops and maintenance begins. One program that illustrates this consideration is the Stryker Armored Vehicle. Formerly known as the Interim Armored Vehicle, the Stryker contract was awarded to General Motors/General Dynamics Land Systems (GM GDLS) in November 2000. GM GDLS was a joint venture between General Motors, Electro-Motive Division and General Dynamics Land Systems Division, specifically created for the sole purpose of developing the Stryker vehicle. The Stryker was envisioned as an interim platform to bridge the gap between legacy armored personnel carriers and a new platform or family of vehicles for FCS. This is evidenced by the acquisition plan of the weapon system, which was not held to normal procurement and production regulations. Because of this, many short-sighted production decisions were made to reduce short term costs. For example, the Stryker initially had only two variants, the Infantry Carrier Vehicle and the Mobile Gun System. By 2007, the Stryker had expanded to 10 different variations, the new ones being the Reconnaissance Vehicle, Mortar Carrier, Commander's Vehicle, Fire Support Vehicle, Anti Tank Guided Missile (ATGM),

Engineer Support Vehicle, medical Evaluation Vehicle, and NBC (Nuclear, Biological, Chemical) Recon Vehicle. It is significant that the Mobile Gun System variant was the second variant envisioned, but the last to be delivered. Such development issues are not uncommon with weapon programs that utilize software dependent advanced fire control systems. One such system is the Army's 155mm self propelled howitzer is the M109A6 Paladin. According to the software support activity for the Stryker the ATGM variant has four times the software as the others, with over 1.2 million SLOC.

Since it was developed as an interim vehicle, the program office did not buy the initial source code from GM GDLS. Software for the Stryker chassis dynamics was written by General Motors Defense. Software for the fire control system was written by General Dynamics. The two contractors had different development teams working with different standards. Complicating matters further, GDLS purchased GMD. Much of the documentation on the chassis dynamics software was lost during this transition. Although they continue to purchase the old code, the software support facilities will likely never acquire the full software build for the Stryker.

The Stryker is still in production. The Army currently has over 1,780 Strykers and the current requirement plans for 2,691 vehicles incorporated into seven Stryker Brigade Combat teams.[36] Some of the variants are five years removed from their initial operating capability. While the program is still receiving full funding for production, undoubtedly software maintenance is being already performed on the various Stryker systems.

The survivability of the interim and objective force's lightweight vehicles (Stryker and FCS) is heavily dependent on information dominance. This has been referred to as "trading armor for information." The ability to achieve real full-spectrum information dominance and lethality cannot be guaranteed, especially within the aggressive deployment schedule of FCS, hence the continued need for continually updated legacy systems.[37]

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D

Sample SPSS Clementine Results

Clementine version 11.1 was used to analyze the validity of the categorical data in the TACOM PPSS spreadsheet. It was determined whether we could simply predict if the budgeted amount fell above or below the median of the data set, from the remaining columns of data. Output from Clementine is depicted in Figure 16. Three models were used, a neural network, a regression tree, and a logistic regression (left, center, and right respectively).

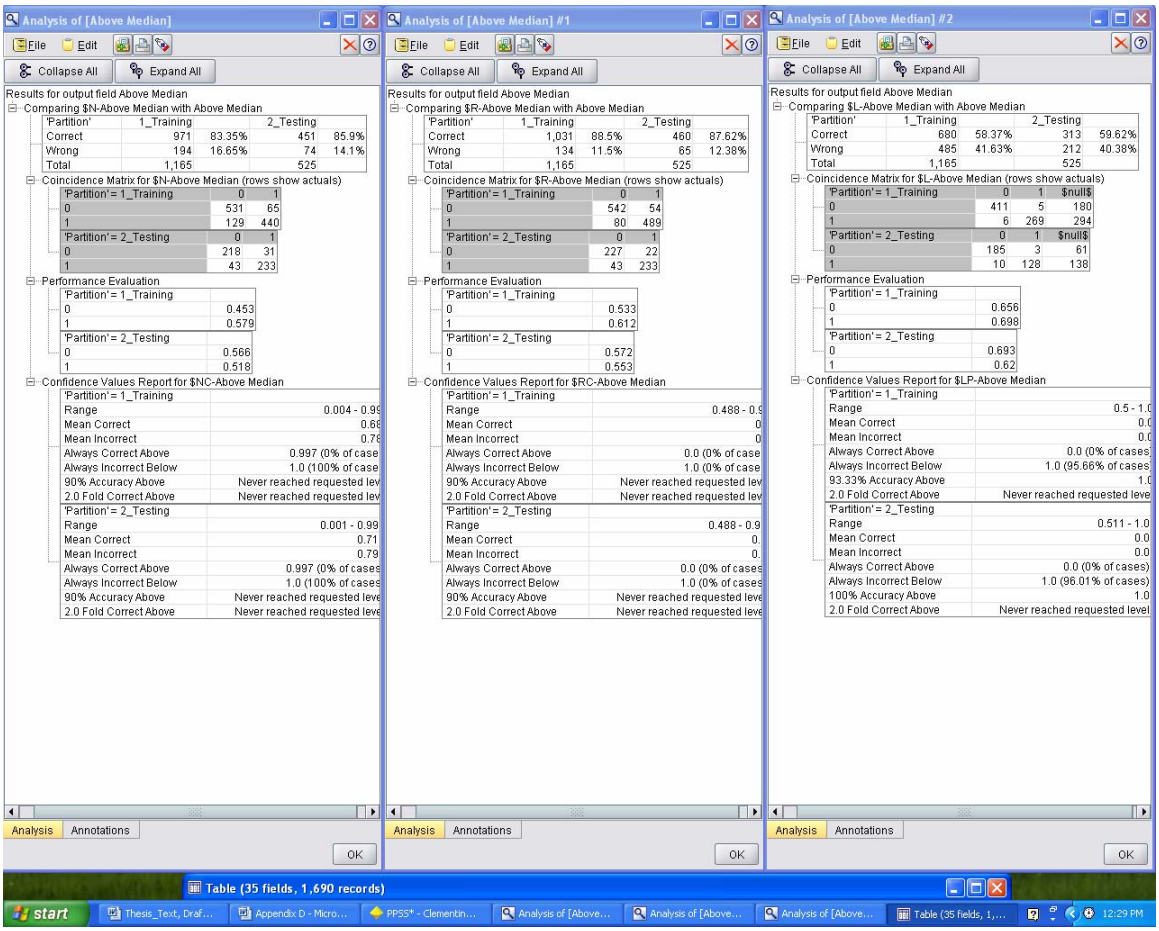


Figure 16. Classification rates of Regression Tree, Neural Network, and Logistic Regression.

The results were as follows. The logistic regression performed almost as badly as the naïve model, which would simply predict that every budgeted amount falls above the median. The simple regression tree actually outperforms the neural network. This suggests that a few variables are a better predictor for the median than a complex formulation. After inspecting the values the regression tree branches to form leaf nodes, it appears that the location of the maintainers, funding, or supported activity. This data is contained in many redundant columns with names such as FIA_WSS_CODE and ORG_CONT_CODE, depicted in Figure 17. This result is supported by an inspection of the relative importance of the neural network variables, where many of them have no appreciable value to the model.

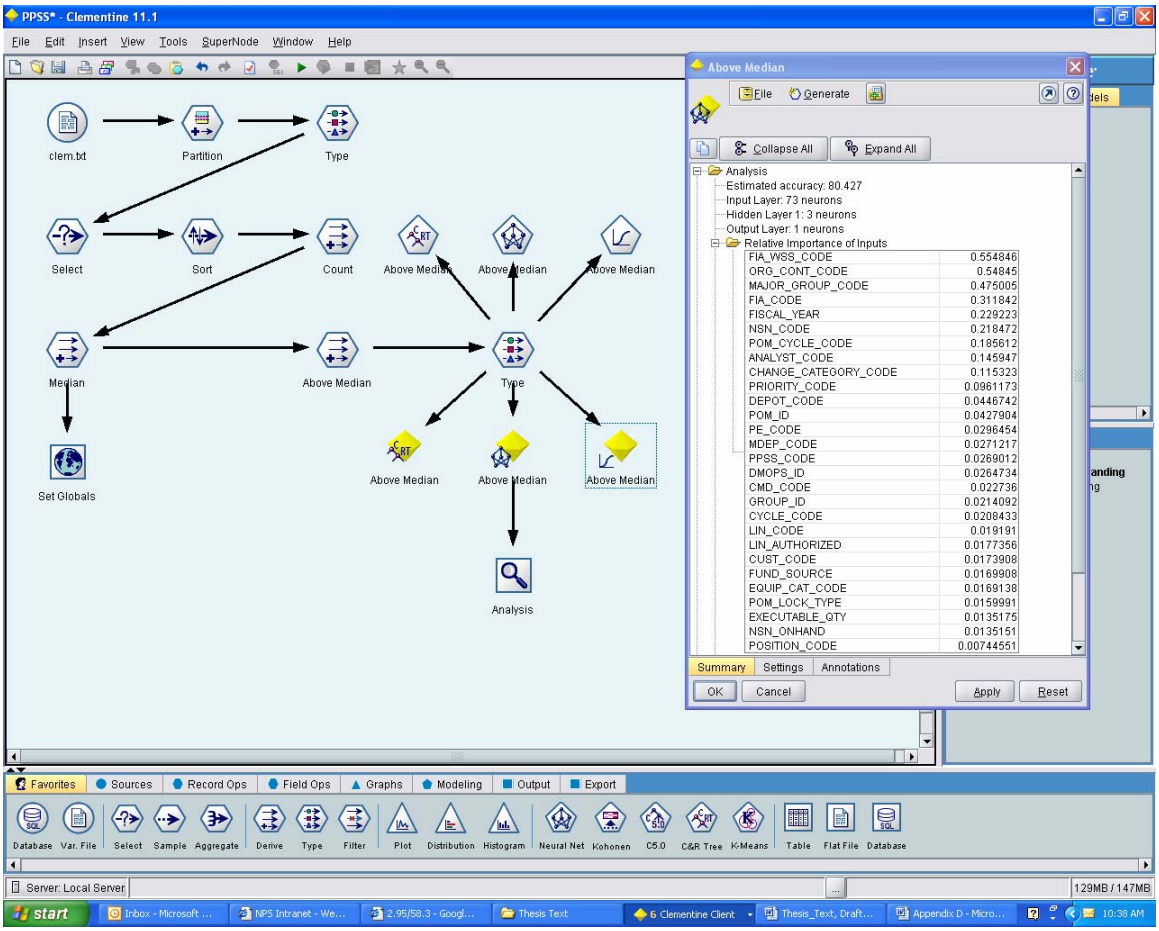


Figure 17. Sample Clementine Stream and Relative Importance of Inputs from the Neural Network.

APPENDIX E

Tables of P-Values from T-Tests

The first table contains p-values comparing annual means between years contained in the data set. No two years are found to be statistically different at the 95% level. The second table compares the compounded percentage change between years. Year pairs that were found to have a greater than 5% change are highlighted.

		2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
	TOTAL	44,220,765	54,469,364	62,961,359	65,248,460	68,706,764	71,000,918	81,046,912	94,438,557	84,995,172	87,050,875	73,998,030	74,922,154
	COUNT	33	45	44	54	57	58	64	71	70	72	54	56
	MEAN	1,340,023	1,210,430	1,430,940	1,208,305	1,205,382	1,224,154	1,266,358	1,330,121	1,214,217	1,209,040	1,370,334	1,337,896
P-VALUES	2002	1.0000	0.7907	0.8675	0.7893	0.7800	0.8063	0.8748	0.9832	0.7823	0.7711	0.9486	0.9963
	2003		1.0000	0.6623	0.9962	0.9908	0.9743	0.8941	0.7779	0.9926	0.9972	0.7052	0.7578
	2004			1.0000	0.6621	0.6512	0.6729	0.7344	0.8366	0.6469	0.6356	0.9010	0.8460
	2005				1.0000	0.9947	0.9706	0.8914	0.7768	0.9885	0.9986	0.7051	0.7570
	2006					1.0000	0.9642	0.8825	0.7649	0.9823	0.9926	0.6913	0.7440
	2007						1.0000	0.9163	0.7942	0.9795	0.9683	0.7177	0.7731
	2008							1.0000	0.8735	0.8911	0.8784	0.7942	0.8539
	2009								1.0000	0.7638	0.7498	0.9205	0.9842
	2010									1.0000	0.9885	0.6842	0.7405
	2011										1.0000	0.6694	0.7257
	2012											1.0000	0.9339
	2013												1.0000
RATES	2002	0.0000	0.1071	-0.0323	0.0351	0.0268	0.0183	0.0095	0.0011	0.0124	0.0115	-0.0022	0.0001
	2003		0.0000	-0.1541	0.0009	0.0014	-0.0028	-0.0090	-0.0156	-0.0004	0.0001	-0.0137	-0.0100
	2004			0.0000	0.1843	0.0896	0.0534	0.0310	0.0147	0.0278	0.0244	0.0054	0.0075
	2005				0.0000	0.0024	-0.0065	-0.0155	-0.0237	-0.0010	-0.0001	-0.0178	-0.0127
	2006					0.0000	-0.0153	-0.0244	-0.0323	-0.0018	-0.0006	-0.0211	-0.0148
	2007						0.0000	-0.0333	-0.0407	0.0027	0.0031	-0.0223	-0.0147
	2008							0.0000	-0.0479	0.0212	0.0156	-0.0195	-0.0109
	2009								0.0000	0.0955	0.0489	-0.0099	-0.0015
	2010									0.0000	0.0043	-0.0587	-0.0318
	2011										0.0000	-0.1177	-0.0494
	2012											0.0000	0.0242
	2013												0.0000

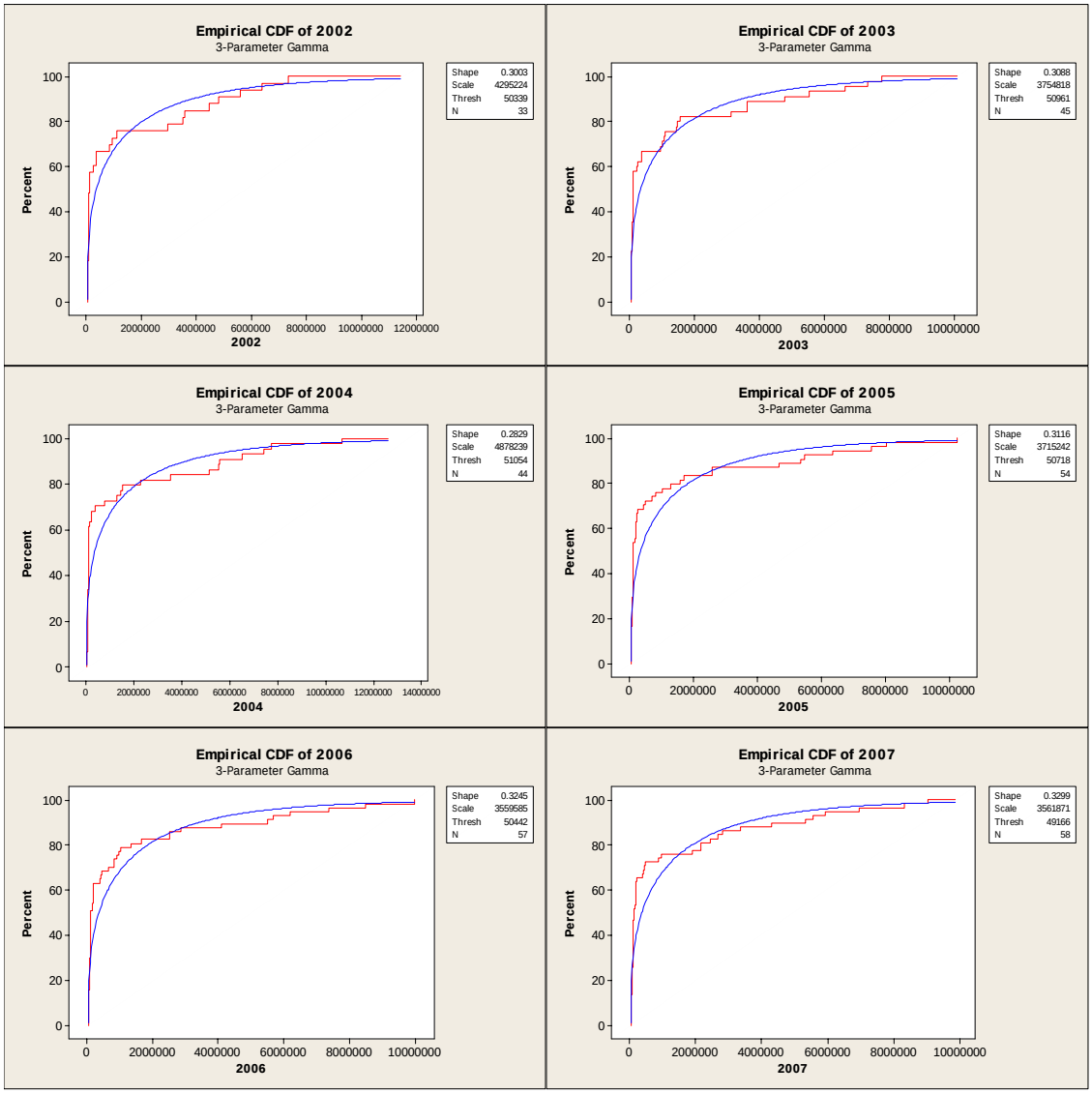
P-Values
Conditional
0.05

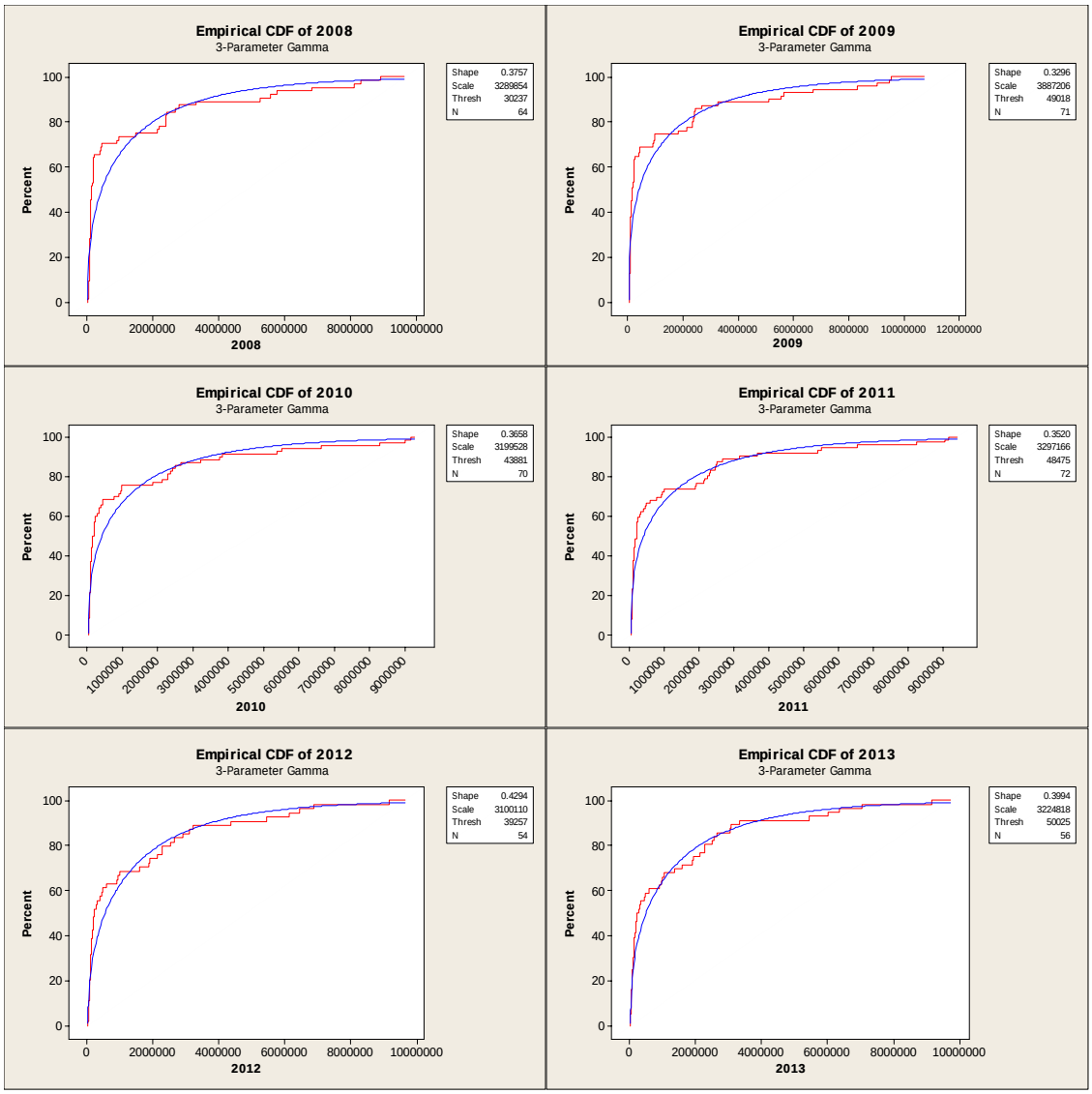
RATES
Conditional
0.05
-0.05

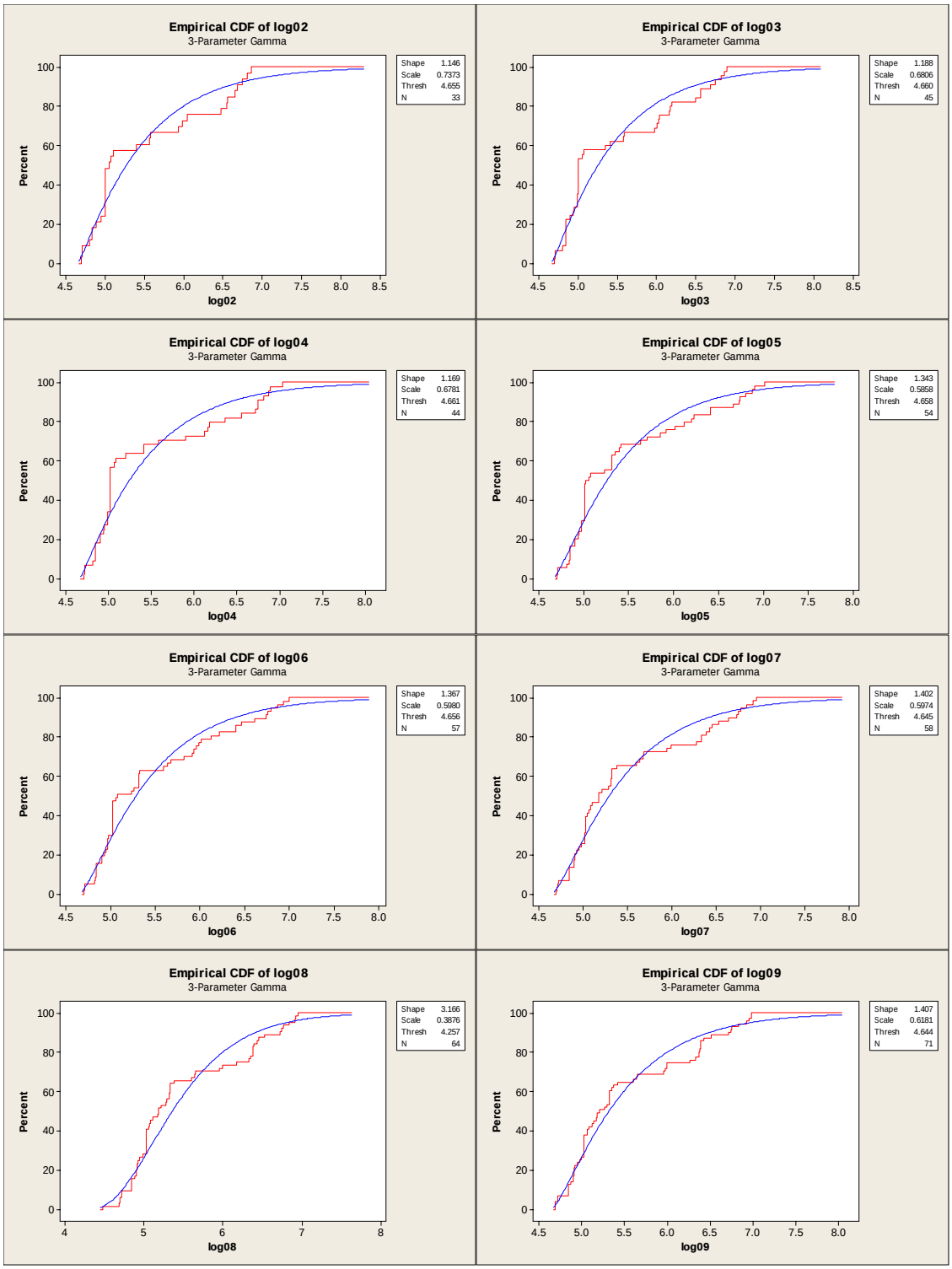
THIS PAGE INTENTIONALLY LEFT BLANK

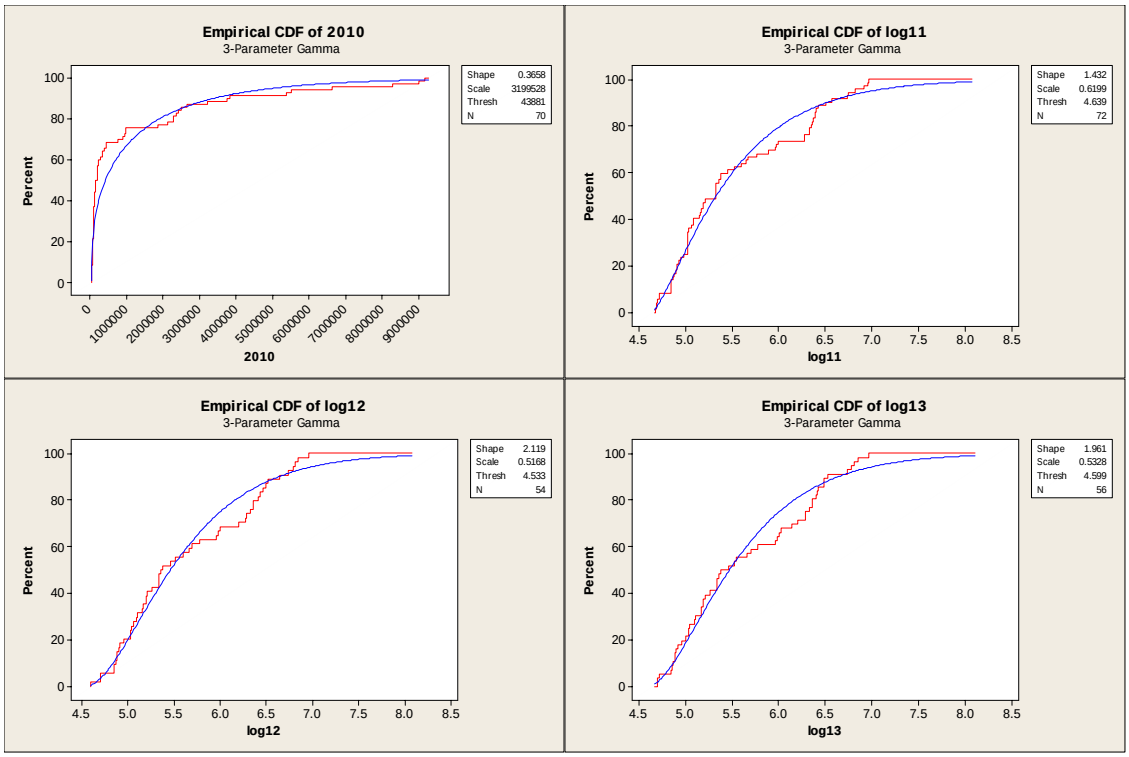
APPENDIX F

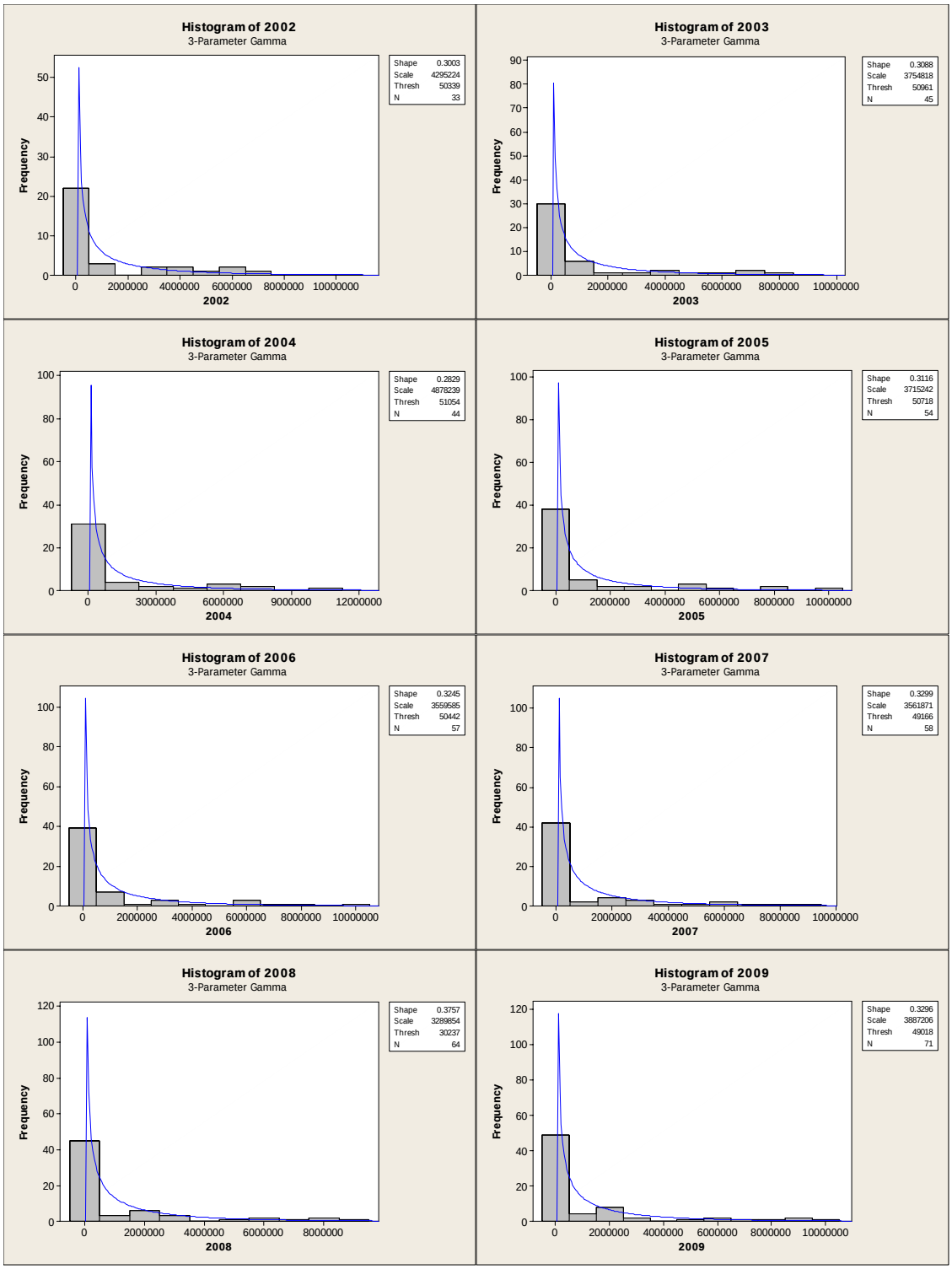
3-Parameter Gamma Distribution Graphs

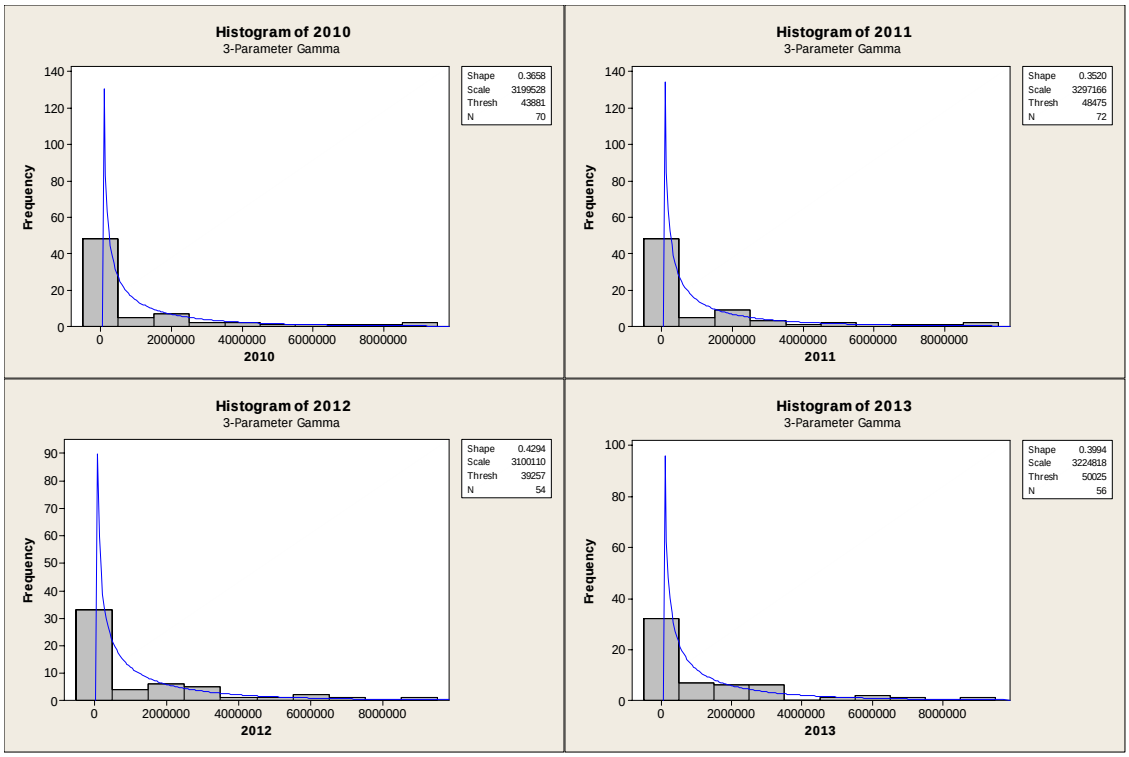


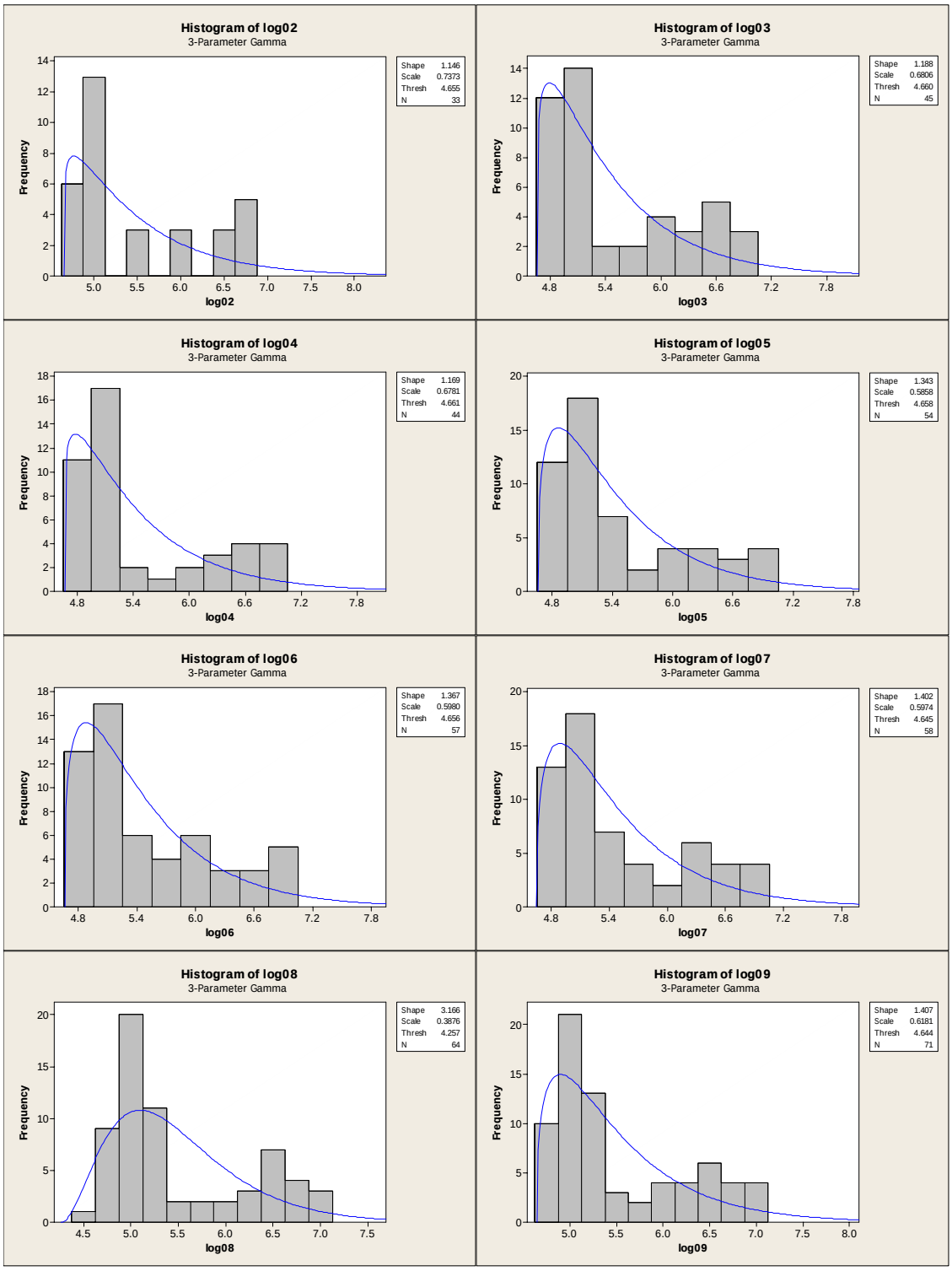

Empirical CDF of 2008
3-Parameter Gamma
Shape 0.3757
Scale 3289854
Thresh 30237
N 64
Percent
2008
Empirical CDF of 2009
3-Parameter Gamma
Shape 0.3296
Scale 3887206
Thresh 49018
N 71
Percent
2009
Empirical CDF of 2010
3-Parameter Gamma
Shape 0.3658
Scale 3199528
Thresh 43881
N 70
Percent
2010
Empirical CDF of 2011
3-Parameter Gamma
Shape 0.3520
Scale 3297166
Thresh 48475
N 72
Percent
2011
Empirical CDF of 2012
3-Parameter Gamma
Shape 0.4294
Scale 3100110
Thresh 39257
N 54
Percent
2012
Empirical CDF of 2013
3-Parameter Gamma
Shape 0.3994
Scale 3224818
Thresh 50025
N 56
Percent
2013

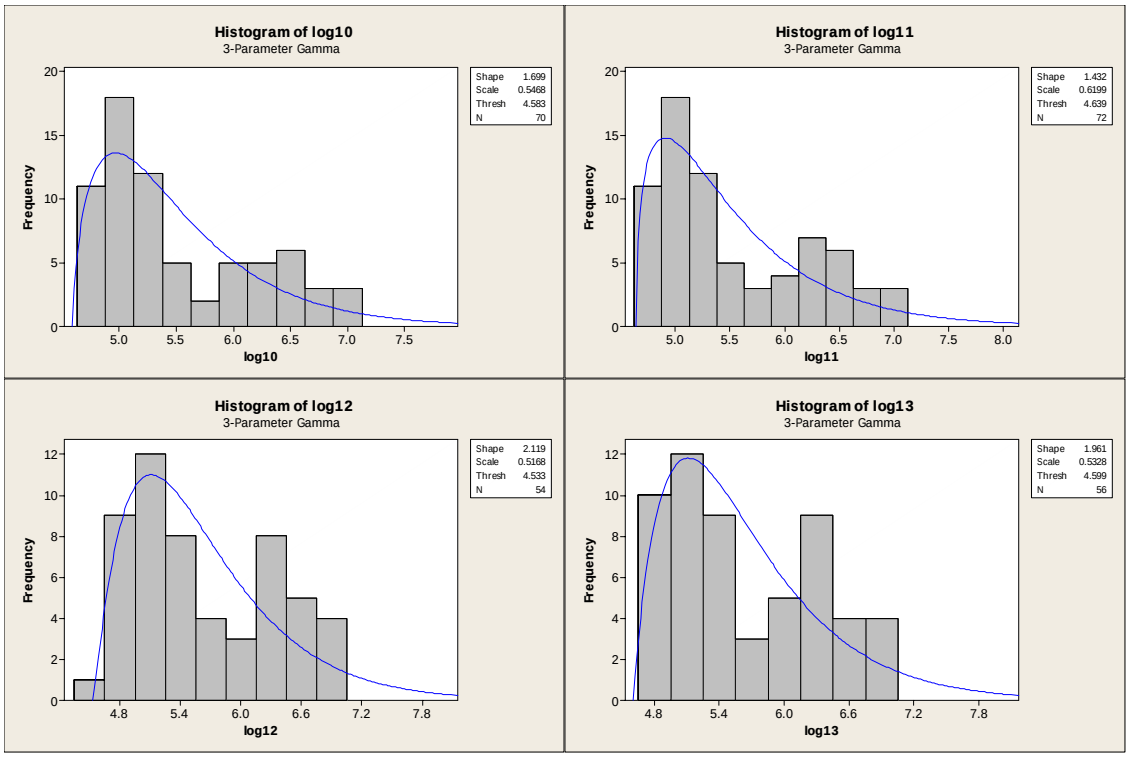

Empirical CDF of log02
3-Parameter Gamma
Shape 1.146
Scale 0.7373
Thresh 4.655
N 33
Percent
log02
Empirical CDF of log03
3-Parameter Gamma
Shape 1.188
Scale 0.6806
Thresh 4.660
N 45
Percent
log03
Empirical CDF of log04
3-Parameter Gamma
Shape 1.169
Scale 0.6781
Thresh 4.661
N 44
Percent
log04
Empirical CDF of log05
3-Parameter Gamma
Shape 1.343
Scale 0.5858
Thresh 4.658
N 54
Percent
log05
Empirical CDF of log06
3-Parameter Gamma
Shape 1.367
Scale 0.5980
Thresh 4.656
N 57
Percent
log06
Empirical CDF of log07
3-Parameter Gamma
Shape 1.402
Scale 0.5974
Thresh 4.645
N 58
Percent
log07
Empirical CDF of log08
3-Parameter Gamma
Shape 3.166
Scale 0.3876
Thresh 4.257
N 64
Percent
log08
Empirical CDF of log09
3-Parameter Gamma
Shape 1.407
Scale 0.6181
Thresh 4.644
N 71
Percent
log09

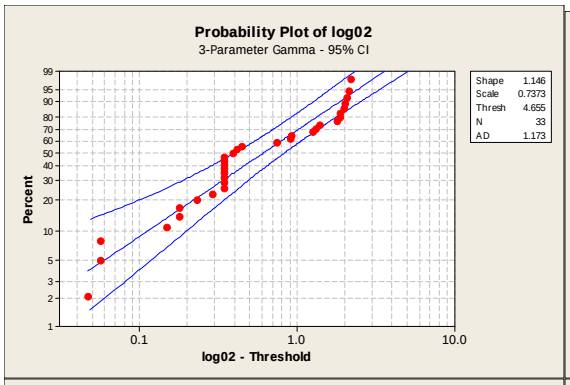

Empirical CDF of 2010
3-Parameter Gamma
Shape 0.3658
Scale 3199528
Thresh 43881
N 70
Percent
2010
Empirical CDF of log11
3-Parameter Gamma
Shape 1.432
Scale 0.6199
Thresh 4.639
N 72
Percent
log11
Empirical CDF of log12
3-Parameter Gamma
Shape 2.119
Scale 0.5168
Thresh 4.533
N 54
Percent
log12
Empirical CDF of log13
3-Parameter Gamma
Shape 1.961
Scale 0.5328
Thresh 4.599
N 56
Percent
log13

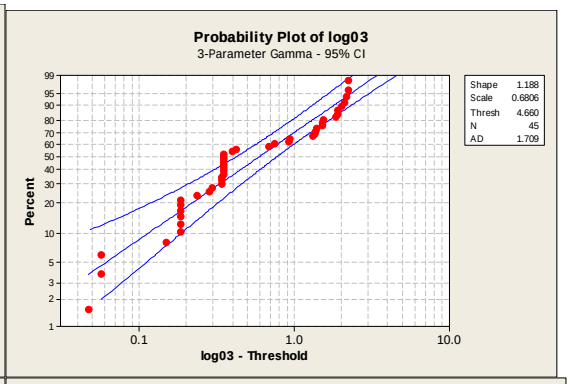

Histogram of 2002
3-Parameter Gamma
Shape 0.3003
Scale 4295224
Thresh 50339
N 33
Frequency
2002
Histogram of 2003
3-Parameter Gamma
Shape 0.3088
Scale 3754818
Thresh 50961
N 45
Frequency
2003
Histogram of 2004
3-Parameter Gamma
Shape 0.2829
Scale 4878239
Thresh 51054
N 44
Frequency
2004
Histogram of 2005
3-Parameter Gamma
Shape 0.3116
Scale 3715242
Thresh 50718
N 54
Frequency
2005
Histogram of 2006
3-Parameter Gamma
Shape 0.3245
Scale 3559585
Thresh 50442
N 57
Frequency
2006
Histogram of 2007
3-Parameter Gamma
Shape 0.3299
Scale 3561871
Thresh 49166
N 58
Frequency
2007
Histogram of 2008
3-Parameter Gamma
Shape 0.3757
Scale 3289854
Thresh 30237
N 64
Frequency
2008
Histogram of 2009
3-Parameter Gamma
Shape 0.3296
Scale 3887206
Thresh 49018
N 71
Frequency
2009

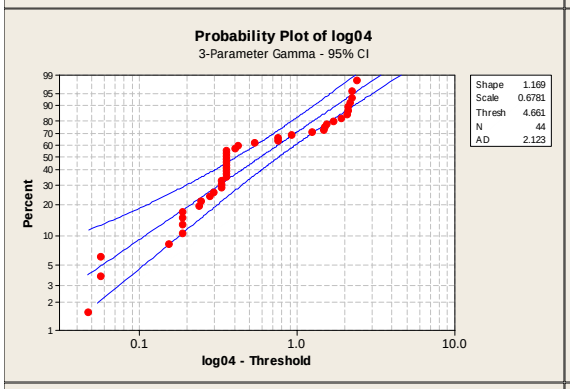

Histogram of 2010
3-Parameter Gamma
Shape 0.3658
Scale 3199528
Thresh 43881
N 70
Frequency
2010
Histogram of 2011
3-Parameter Gamma
Shape 0.3520
Scale 3297166
Thresh 48475
N 72
Frequency
2011
Histogram of 2012
3-Parameter Gamma
Shape 0.4294
Scale 3100110
Thresh 39257
N 54
Frequency
2012
Histogram of 2013
3-Parameter Gamma
Shape 0.3994
Scale 3224818
Thresh 50025
N 56
Frequency
2013

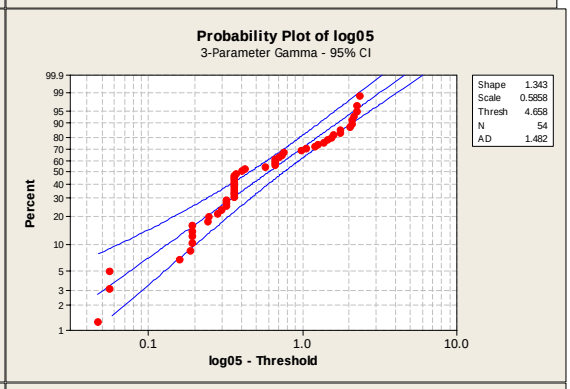

Histogram of log02
3-Parameter Gamma
Shape 1.146
Scale 0.7373
Thresh 4.655
N 33
Frequency
log02
Histogram of log03
3-Parameter Gamma
Shape 1.188
Scale 0.6806
Thresh 4.660
N 45
Frequency
log03
Histogram of log04
3-Parameter Gamma
Shape 1.169
Scale 0.6781
Thresh 4.661
N 44
Frequency
log04
Histogram of log05
3-Parameter Gamma
Shape 1.343
Scale 0.5858
Thresh 4.658
N 54
Frequency
log05
Histogram of log06
3-Parameter Gamma
Shape 1.367
Scale 0.5980
Thresh 4.656
N 57
Frequency
log06
Histogram of log07
3-Parameter Gamma
Shape 1.402
Scale 0.5974
Thresh 4.645
N 58
Frequency
log07
Histogram of log08
3-Parameter Gamma
Shape 3.166
Scale 0.3876
Thresh 4.257
N 64
Frequency
log08
Histogram of log09
3-Parameter Gamma
Shape 1.407
Scale 0.6181
Thresh 4.644
N 71
Frequency
log09

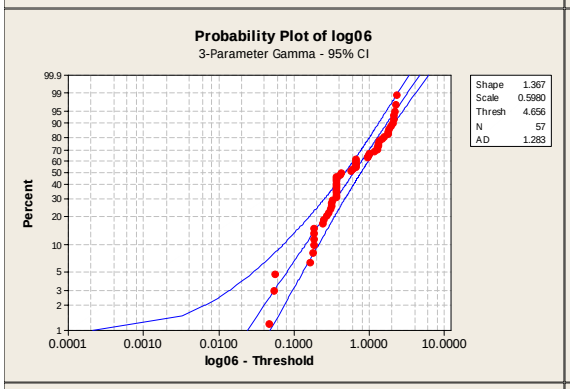

Histogram of log10
3-Parameter Gamma
Shape 1.699
Scale 0.5468
Thresh 4.583
N 70
Frequency
log10
Histogram of log11
3-Parameter Gamma
Shape 1.432
Scale 0.6199
Thresh 4.639
N 72
Frequency
log11
Histogram of log12
3-Parameter Gamma
Shape 2.119
Scale 0.5168
Thresh 4.533
N 54
Frequency
log12
Histogram of log13
3-Parameter Gamma
Shape 1.961
Scale 0.5328
Thresh 4.599
N 56
Frequency
log13

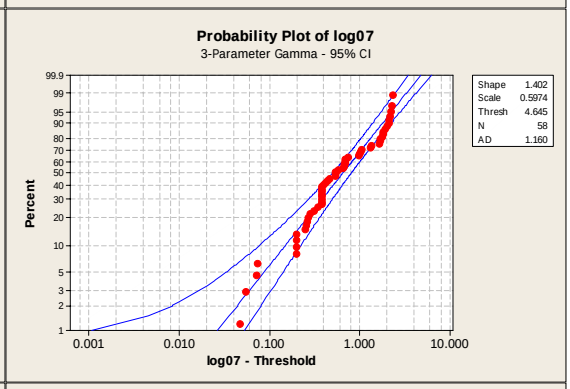

Probability Plot of log02
3-Parameter Gamma - 95% CI
Shape 1.146
Scale 0.7373
Thresh 4.655
N 33
AD 1.173
Percent
log02 - Threshold

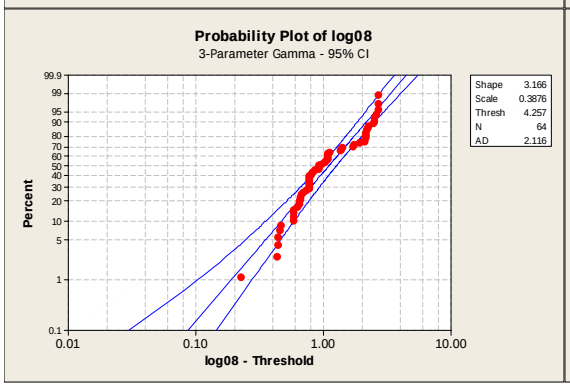

Probability Plot of log03
3-Parameter Gamma - 95% CI
Shape 1.188
Scale 0.6806
Thresh 4.660
N 45
AD 1.709
Percent
log03 - Threshold

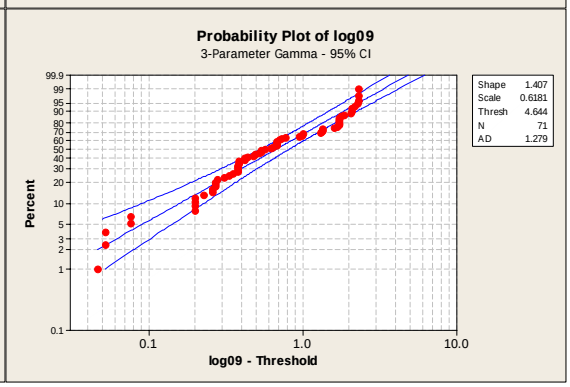

Probability Plot of log04
3-Parameter Gamma - 95% CI
Shape 1.169
Scale 0.6781
Thresh 4.661
N 44
AD 2.123
Percent
log04 - Threshold

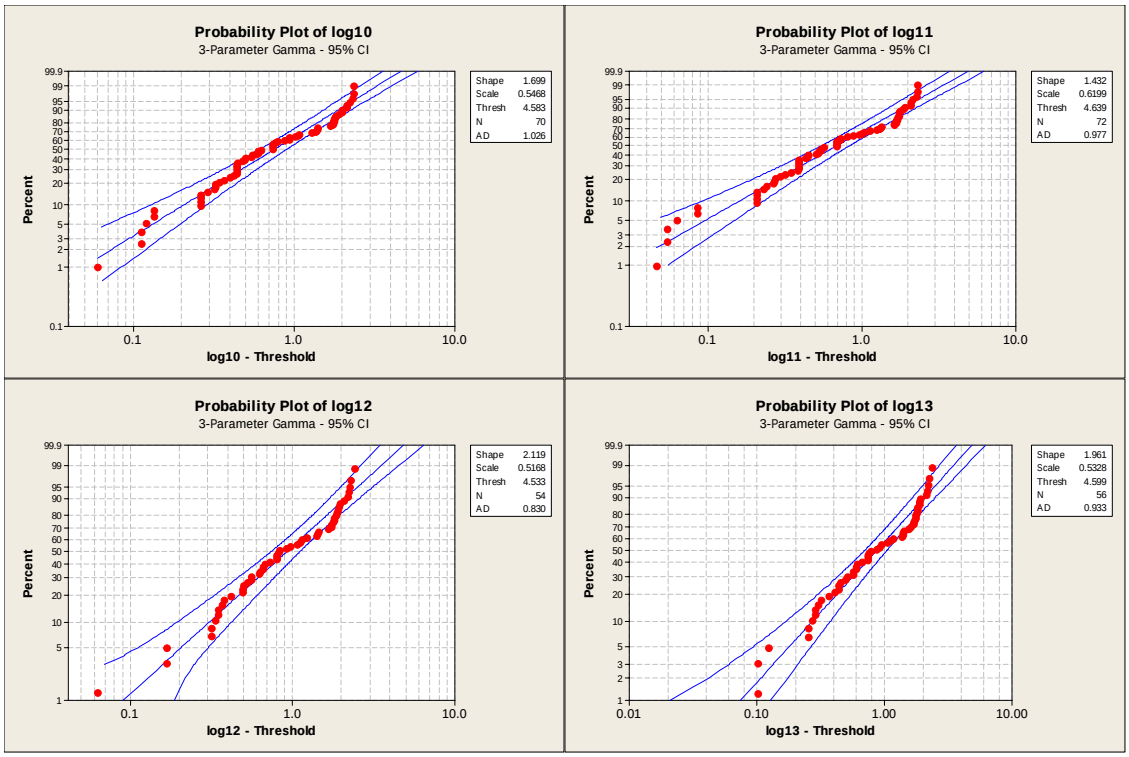

Probability Plot of log05
3-Parameter Gamma - 95% CI
Shape 1.343
Scale 0.5858
Thresh 4.658
N 54
AD 1.482
Percent
log05 - Threshold

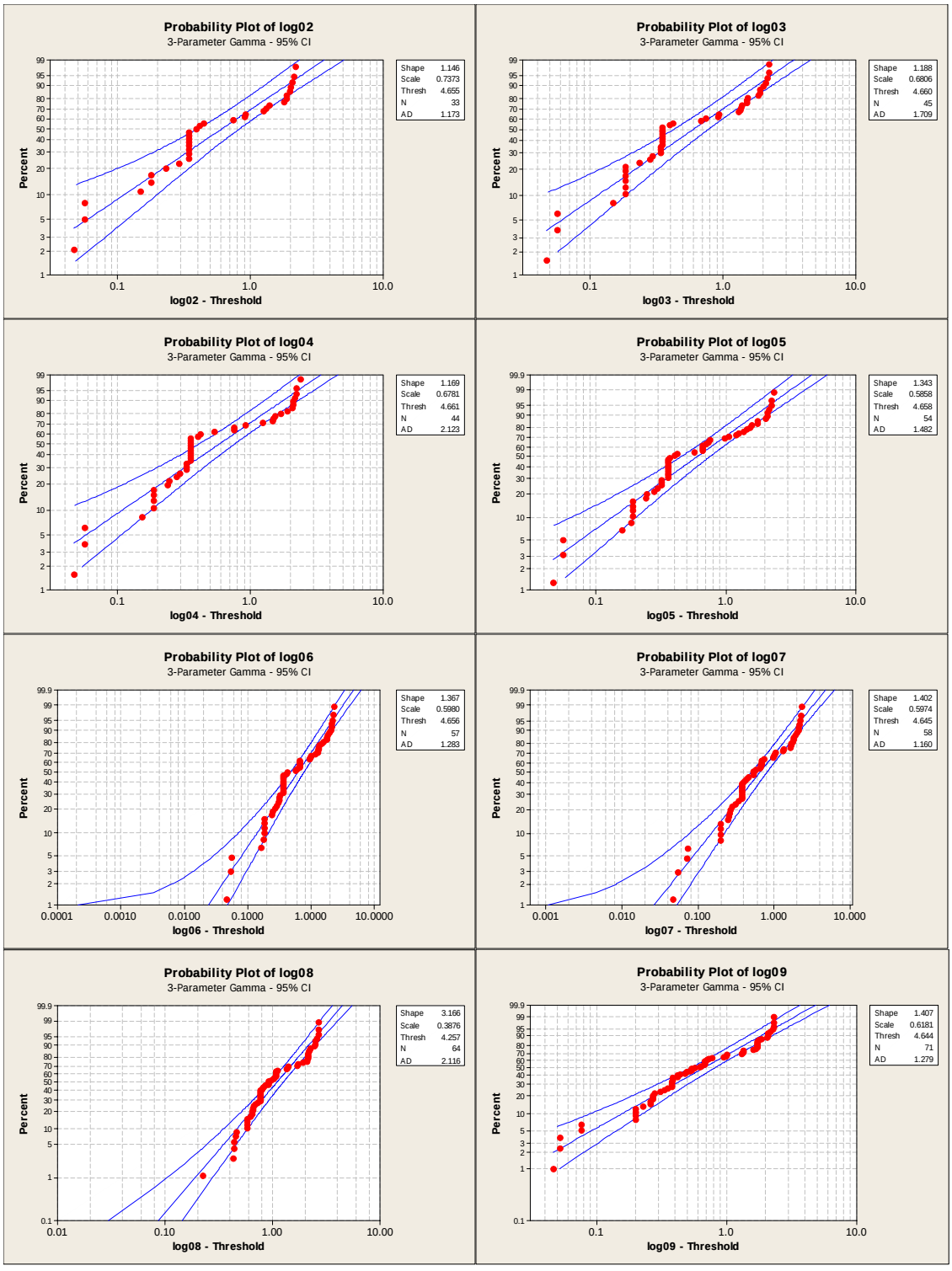

Probability Plot of log06
3-Parameter Gamma - 95% CI
Shape 1.367
Scale 0.5980
Thresh 4.656
N 57
AD 1.283
Percent
log06 - Threshold

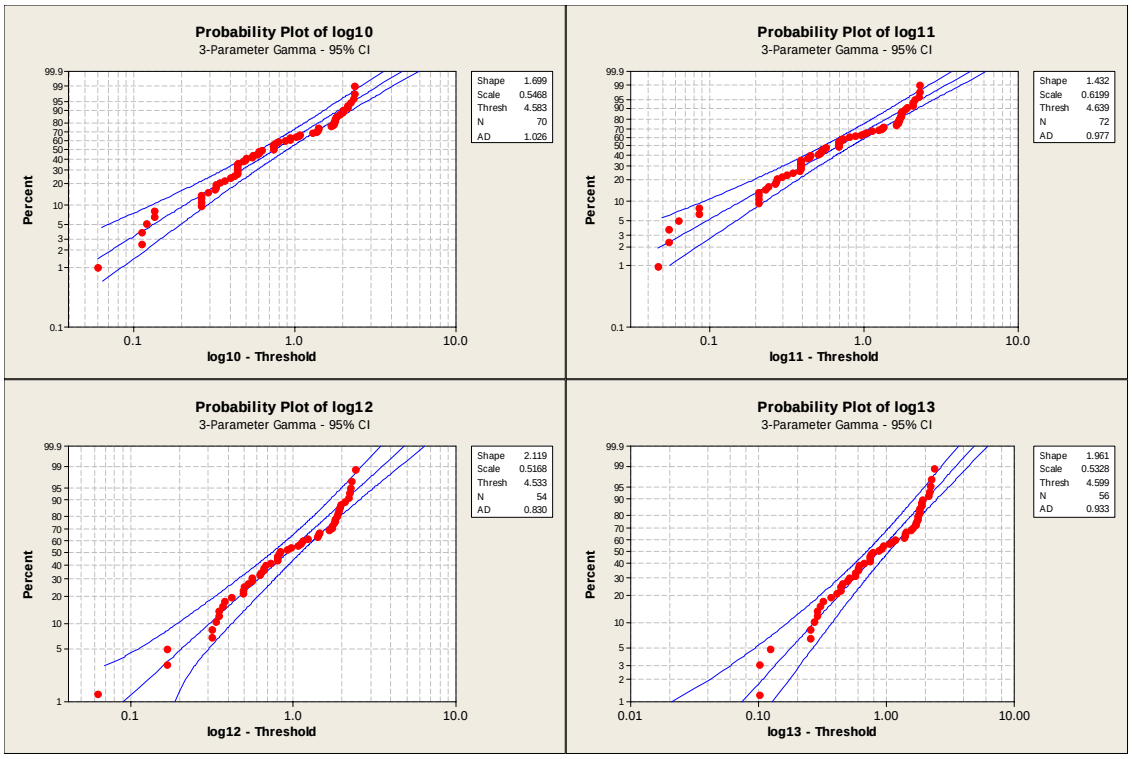

Probability Plot of log07
3-Parameter Gamma - 95% CI
Shape 1.402
Scale 0.5974
Thresh 4.645
N 58
AD 1.160
Percent
log07 - Threshold


Probability Plot of log08
3-Parameter Gamma - 95% CI
Shape 3.166
Scale 0.3876
Thresh 4.257
N 64
AD 2.116
Percent
log08 - Threshold


Probability Plot of log09
3-Parameter Gamma - 95% CI
Shape 1.407
Scale 0.6181
Thresh 4.644
N 71
AD 1.279
Percent
log09 - Threshold


Probability Plot of log10
3-Parameter Gamma - 95% CI
Shape 1.699
Scale 0.5468
Thresh 4.583
N 70
AD 1.026
Percent
log10 - Threshold
Probability Plot of log11
3-Parameter Gamma - 95% CI
Shape 1.432
Scale 0.6199
Thresh 4.639
N 72
AD 0.977
Percent
log11 - Threshold
Probability Plot of log12
3-Parameter Gamma - 95% CI
Shape 2.119
Scale 0.5168
Thresh 4.533
N 54
AD 0.830
Percent
log12 - Threshold
Probability Plot of log13
3-Parameter Gamma - 95% CI
Shape 1.961
Scale 0.5328
Thresh 4.599
N 56
AD 0.933
Percent
log13 - Threshold


Probability Plot of log02
3-Parameter Gamma - 95% CI
Shape 1.146
Scale 0.7373
Thresh 4.655
N 33
AD 1.173
Percent
log02 - Threshold
Probability Plot of log03
3-Parameter Gamma - 95% CI
Shape 1.188
Scale 0.6806
Thresh 4.660
N 45
AD 1.709
Percent
log03 - Threshold
Probability Plot of log04
3-Parameter Gamma - 95% CI
Shape 1.169
Scale 0.6781
Thresh 4.661
N 44
AD 2.123
Percent
log04 - Threshold
Probability Plot of log05
3-Parameter Gamma - 95% CI
Shape 1.343
Scale 0.5858
Thresh 4.658
N 54
AD 1.482
Percent
log05 - Threshold
Probability Plot of log06
3-Parameter Gamma - 95% CI
Shape 1.367
Scale 0.5980
Thresh 4.656
N 57
AD 1.283
Percent
log06 - Threshold
Probability Plot of log07
3-Parameter Gamma - 95% CI
Shape 1.402
Scale 0.5974
Thresh 4.645
N 58
AD 1.160
Percent
log07 - Threshold
Probability Plot of log08
3-Parameter Gamma - 95% CI
Shape 3.166
Scale 0.3876
Thresh 4.257
N 64
AD 2.116
Percent
log08 - Threshold
Probability Plot of log09
3-Parameter Gamma - 95% CI
Shape 1.407
Scale 0.6181
Thresh 4.644
N 71
AD 1.279
Percent
log09 - Threshold


Probability Plot of log10
3-Parameter Gamma - 95% CI
Shape 1.699
Scale 0.5468
Thresh 4.583
N 70
AD 1.026
Percent
log10 - Threshold
Probability Plot of log11
3-Parameter Gamma - 95% CI
Shape 1.432
Scale 0.6199
Thresh 4.639
N 72
AD 0.977
Percent
log11 - Threshold
Probability Plot of log12
3-Parameter Gamma - 95% CI
Shape 2.119
Scale 0.5168
Thresh 4.533
N 54
AD 0.830
Percent
log12 - Threshold
Probability Plot of log13
3-Parameter Gamma - 95% CI
Shape 1.961
Scale 0.5328
Thresh 4.599
N 56
AD 0.933
Percent
log13 - Threshold

LIST OF REFERENCE

1. "The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition" pages 25-26, Fredrick P. Brooks, Jr. Reading, MA: Addison-Wesley, 1995.

2. Open Architecture (OA) Computing Environment Design Guidance Version 1.0 (August 23, 2004) page 45 Retrieved July 2007 http://www.nswc.navy.mil/wwwDL/B/OACE/docs/OACE_Design_Guidance_v1dot0_final.pdf.

3. Personal Visit, TARDEC Next Generation Software Lab December 21, 2006.

4. "Tactical Aircraft: DoD Should present a New F-22A Business Case before Making Further Investments," David M. Walker, Comptroller General of the U.S. June 20, 2006 Retrieved July 2007 http://www.cdi.org/pdfs/GAO%20on%20F%2022%20per%20June%202006.pdf.

5. "Status of the F/A-22 and JSF Acquisition Programs and Implications for Tactical Aircraft Modification," Michael Sullivan and Allen Li, March 3, 2006. Retrieved July 2007 http://www.gao.gov/new.items/d05390t.pdf.

6. Communications of the ACM 30(5) 416-429. "*An Empirical Validation of Software Cost Estimation Models*." Kemerer, C.F. (1987).

7. **"**Software Metrics: A Rigorous and Practical Approach" Fenton, N.E. and Pfleeger, S.L. (1997). International Thomson Computer Press.

8. **"**Software Metrics: A Rigorous and Practical Approach" Fenton, N.E. and Pfleeger, S.L. (1997). International Thomson Computer Press.

9. GAO report "The Army's Future Combat Systems' Features, Risks, and Alternatives" Paul L. Francis, April 1, 2004. Retrieved July 2007 http://www.gao.gov/new.items/d04635t.pdf.

10. "Rainfall Frequency Analysis Using a Mixed Gamma Distribution: Evaluation of the Global Warming Effect on Daily Rainfall." Chulsang Yoo, Kwang-Sik Jung, Tae-Woong Kim. Originally published online in Wiley Interscience (www.interscience.wiley.cocm). Retrieved August 2007 http://www3.interscience.wiley.com/cgi-bin/fulltext/112159729/PDFSTART?CRETRY=1&SRETRY=0.

11. Table reproduced from: "How CMM Rating Impacts Quality, Productivity, Rework, and the Bottom Line" STSC CrossTalk, Jeff King and Michael Diaz, March 2002. Retrieved August 2007 http://www.stsc.hill.af.mil/crosstalk/2002/03/diaz.html.

12. Proceedings of the First International Research Workshop of Process Improvement in Small Settings, 2005 Suzanne Garcia, Caroline Graettinger, Keith Kost, Pages109-113. Retrieved August 2007 http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06sr001.pdf.

13. Similar recommendations were previously suggested by Brian D. Fersch, Operations Research Analyst, Office of the Deputy Assistant Secretary of the Army for Cost & Economics (ODASA-CE); Networks, Information, Software, and Electronics Costing Division Software Cost.

14. July 2007 correspondence with James M. Judy, Chief, Networks, Information, Software & Electronics Costing (NISEC) Division; ODASA-CE.

15. "Denver Airport Saw the Future. It Didn't Work" Kirk Johnson, New York Times, published August 27, 2005. Retrieved July 2007 http://www.nytimes.com/2005/08/27/national/27denver.html?ex=1282795200&en=55c1a4d8ddb7988a&ei=5088&partner=rssnyt&emc=rss.

16. "Famous Failures of Complex Engineering Systems," Retrieved July 2007 http://www.cds.caltech.edu/conferences/1997/vecs/tutorial/Examples/Cases/failures.htm.

17. "They Write the Right Stuff" Charles Fishman, Issue 6, page 95 December 1996. Retrieved July 2007 http://www.fastcompany.com/online/06/writestuff.html.

18. "Lockheed Martin Receives $61 Million Contract for Multifunction Utility/Logistics and Equipment Vehicle" Press release, Craig Vanbebber, June 22, 2005. Retrieved July 2007 http://www.lockheedmartin.com/wms/findPage.do?dsp=fec&ci=16902&rsbci=0&fti=112&ti=0&sc=400.

19. "Manned and Unmanned Ground Vehicle ISR Sensors Announcement," Press Release, Howard Lind, February 8, 2006. Retrieved July 2007 http://www.boeing.com/defense-space/ic/fcs/bia/060201_ems_rfi.html.

20. "Auburn University Awarded Subcontract from FCS LSI," Press Release, Sara Borchik, September 21, 2006. Retrieved July 2007 http://eng.auburn.edu/admin/marketing/newsroom/2006/september/ausubcontract.html.

21. "General Dynamics Awarded Additional FCS Command and Control Integration Work," Press Release, Fran Jacques April 21, 2005. Retrieved July 2007 http://www.gdc4s.com/news/detail.cfm?prid=168.

22. BAE Systems Armament Systems Division, retrieved July 2007 http://www.na.baesystems.com/landArmaments.cfm.

23. "Curtiss-Wright Awarded Contract for Future Combat System," Press Release, Alexandra Deignan October 2, 2006. Retrieved July 2007 http://phx.corporate-ir.net/phoenix.zhtml?c=81495&p=irol-newsArticle&ID=910891&highlight=.

24. "General Atomics Lands $10.7 Million R&D Contract to Develop Future Power Systems" Press Release February 12, 2007. Retrieved July 2007 http://www.defenseindustrydaily.com/general-atomics-lands-107-million-rd-contract-to-develop-future-power-systems-03039/.

25. "iRobot's Future Combat Systems Contract Grows to Over $41 Million," Press Release Parna Sarkar May 31, 2005. Retrieved July 2007 http://www.irobot.com/sp.cfm?pageid=86&id=147&referrer=85.

26. "Boeing and SAIC Award Honeywell Contract to Develop Future Combat System Class I Unmanned Aerial Vehicles," Press Release Mary McAdam May 24, 2006. Retrieved August 2007 http://www.boeing.com/ids/news/2006/q2/060524a_nr.html.

27. "G Systems delivers NLOS-LS Test Set to Lockheed Martin" Press Release Andrew Kahn, April 5, 2007. Retrieved July 2007 http://www.gsystems.com/industries/G%20Systems%20NLOS%20delivery_04_05_07.pdf.

28. "FCS Partner Selections" News Release 2003. Retrieved July 2007 http://www.boeing.com/news/releases/2003/q3/nr_030710m_list.html.

29. "C4ISR Architecture and tactical Systems Planning Tool" SITIS Archives. Retrieved July. 2007 http://www.dodsbir.net/Sitis/archives_display_topic.asp?Bookmark=10830.

30. "Training Common Components" U.S. Army PEO STRI. Retrieved July 2007 http://www.peostri.army.mil/PRODUCTS/TCC/.

31. "Ada Joint Programming Office Mission Accomplished, 1980-1998 FAQ" Retrieved July 2007 http://sw-eng.falls-church.va.us/AdaIC/.

32. "F-22 Software Risk Reduction," STSC, CrossTalk Beverly L. Moody, F-22 Avionics Software Block lead, May 2000. Retrieved June 2007 http://www.stsc.hill.af.mil/Crosstalk/2000/05/moody.html.

33. "F/A-22 Program History" Retrieved July 2007 http://www.f-22raptor.com/index_airframe.php#1992\.

34. Murphy's Law article April 2, 2004. Retrieved June 2007 http://www.strategypage.com/htmw/htmurph/articles/20040402.aspx.

35. "Tactical Aircraft: DOD Should present a New F-22A Business Case before Making Further Investments," David M. Walker, Comptroller General of the U.S. June 20, 2006 Retrieved July 2007 http://www.cdi.org/pdfs/GAO%20on%20F%2022%20per%20June%202006.pdf.

36. "Stryker 8-Wheel Drive Armored Combat Vehicles" Army-technology.com, retrieved July 2007 http://www.army-technology.com/projects/stryker/.

37. "Land Combat Systems Industry Report Academic Year 2002-2003" LTC Michael E. Donovan et al. Retrieved July 2007 http://www.ndu.edu/icaf/industry/IS2003/papers/2003%20Land%20Combat%20Systems.htm.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Commanding General, Training and Education Command
   MCCDC, Code C46
   Quantico, Virginia

4. Director, Marine Corps Research Center
   MCCDC, Code C40RC
   Quantico, Virginia

5. Marine Corps Tactical Systems Support Activity
   (Attn: Operations Officer)
   Camp Pendleton, California

6. Director, Operations Analysis Division
   Code C19, MCCDC
   Quantico, Virginia

7. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

8. MCCDC OAD Liaison to Operations Research Department
   Naval Postgraduate School
   Monterey, California

9. Walt Cooper
   OSD, PA&E/CAIG
   The Pentagon, 2D278

10. Tom Ogilvy
    Army G-4, Resource Integration Division
    The Pentagon, 1D343-16